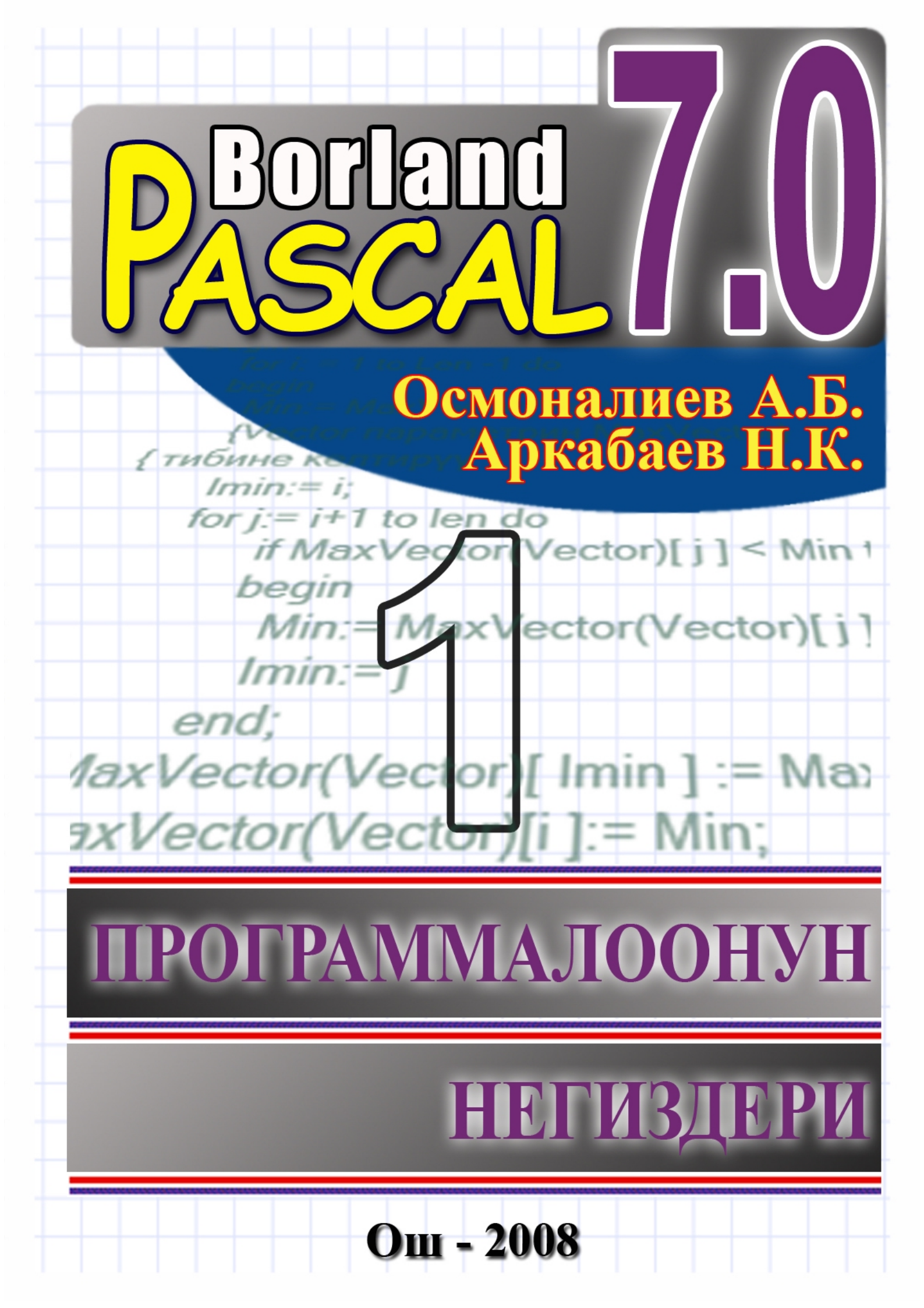
Borland PASCAL 7.0

Осмоналиев А.Б.
Аркабаев Н.К.



1

ПРОГРАММАЛООНУН НЕГИЗДЕРИ

Ош - 2008

МАЗМУНУ

АЛГЫ СӨЗ

Башка тилди жандай жакшы көрсөм да,
Эне тилди сүйгөнүмдөн жаңылбайм.
(Алыкул Осмонов)

Азыркы учурда персоналдык компьютерлер илимдин жана эл чарбасынын өнүгүшүндө дегеле адамзаттын турмушунда чечүүчү факторлордун бири болуп калды. Компьютерлердин адамзат турмушунун ар түрдүү сфераларына сүңгүп кириши алардын көптөгөн актуалдуу маселелерди чечүүдө колдонулушун шарттоо менен дээрлик ар бир жумушчу орунда компьютерлерди колдонуу зарылдыгына алып келди. Компютердик техникалар жаатындагы прогресс келечек өз ишмердүүлүгүндө компьютердик илимдердин жетишкендиктерин сабаттуу колдоно билген инсандардыкы болорун айгинелеп олтурат.

Ушундай информациялык доордун эпкининен четте калбаган биздин эгемендүү Кыргыз Республикасында да албетте, акыркы он жылдарда бир топ кубанарлык жетишкендиктер болду. Көптөгөн окуу жайларыбызда, мектептерде, ишкана-мекемелерде ал тургай балдар бакчаларына чейин дээрлик акыркы үлгүдөгү компьютерлер орун алды.

Көптөгөн жогорку окуу жайларыбызда ушул компьютердик техникаларды иштете билүүчү, эң оболу компьютерлердин программалык камсыздандыруусу менен түздөн-түз иш алып баруучу адистер даярдалынып келүүдө. Бул адистиктердин окуу пландарында компьютердик илимдер тобуна кирген бир нече окуу предметтери каралган. Окутуу тажрыйбасы көрсөткөндөй, компьютердик илимдер боюнча предметтерди окуп үйрөнүүдө окуучуларыбыз, студенттерибиз жана окутуучулар биз өзүбүз да өңчөй орус же англис тилинде жазылган адабияттардан пайдаланып келебиз. Эми окутууну кыргыз тилинде алып баруу үчүн мындагы көптөгөн терминдерди же аталыштарды кыргыз тилине которуп жеткирүү зарылчылыгы пайда болот. Алибетте мындай учурда аларды четтен эле которо берүү мүмкүн эмес жана ага аракет жасоонун өзү акылсыздык болмок. Ошондой болсо да көпчүлүк аталыштар үчүн кыргыз тилиндеги алардын эң сонун эквиваленттерин табууга болот.

Дагы бир белгилеп кетчүү нерсе, ушундай предметтерди алып баруучу окутуучуларыбыздын арасында өз ара кандайдыр бир макулдашуучулуктун жоктугунан улам компьютердик илимдер боюнча аталыштар жана сөз байланыштары ар түрдүү авторлор тарабынан ар түрдүүчө которулуп жүрөт. Бул болсо ар түрдүү

булактардан пайдалануу зарыл болгон учурда кандайдыр бир деңгээлде чаташууларды пайда кылат.

Үчүнчү бир жагдай, эгемендүүлүккө жетишкендиктен бери, кыргыз тили ушул эгемендүү мамлекеттеги малекеттик тил катары жарыялангандан бери бир топ жыл өтсө да компьютердик илимдер боюнча кыргызча басылмалар, анча-мынча китепчелерди жана бирин-экин китептерди эске албаганда, дээрлик жокко эсе. Ошол эле учурда кошуна мамлекеттерде бул иш бир топ жакшы бараткандыгына күбө болуп жүрөбүз.

Биринчиден ушул жагдайлардан улам, экинчи жактан студенттердин суроо-талаптары эске алынып ушул китеп жазылып олтурат. Албетте, мында келтирилген котормолор идеалдуу болду дегенден алыспыз, бирок керектүү материалдын маанисин бузбай бере алгыдай болду деп эсептейбиз. Эгер мындагы котормолор үчүн башка варианттар сизге белгилүү болуп бизге билдирсеңиздер, анда аларды биз чоң ыраазычылык менен кабыл алар элек.

Китеп 12 баптан туруп Borland Pascal 7.0 тилинин чөйрөсүндө программалоо курсунун биринчи бөлүгү болуп эсептелет. Мында Borland Pascal 7.0 тилинин чөйрөсүндө динамикалык берилгендер менен иштөө, объекттик-ориентирдик программалоо, графика жана стандарттык түзүлүштөр менен иштөөгө чейинки материалдар берилди. Ар бир бап келтирилген материалды жеткилең түшүнүүгө өбөлгө болуучу жетишерлик сандагы мисалдар менен камсыз болгон жана акырында өз алдынча иштөөгө карата мисалдардын тобу келтирилген. Мындагы келтирилген мисалдардын бир тобу ар түрдүү булактардан алардын абдан демонтрациялуулугунан улам кээ бир өзгөртүүлөр менен пайдаланылды. Аларга биз автор болууга эч кандай акыбыз жок. Андан сырткары компилятордун каталар жөнүндөгү билдирүүлөрүнүн жана компилятор үчүн директивалардын тизмеси толук келтирилди.

Китеп мектептердин жогорку класстарынын окуучулары, жогорку окуу жайлардын студенттери жана жалпы эле Borland Pascal 7.0 тилинин чөйрөсүндө программалоону үйрөнүүнү каалагандар үчүн арналып жазылды.

Бул окуу китеп боюнча окурмандардын суроолору, сын-пикирлери жана каалоолору чоң ыраазычылык менен төмөнкү даректер боюнча кабыл алынат: okamil1@rambler.ru, nike_osh@rambler.ru, nurkasym@gmail.com.

Авторлор

КИРИШҮҮ

Айт айт десе ааламды айт,
Аалам алга кадамдайт.
(Барпы)

Белгилүү француз математиги жана философу Блэз Паскальдын (1623-1662) урматына анын аты менен аталган Паскаль программалоо тилинин пайда болушу XX кылымдын 70-жылдарына тура келет. Бул тил Цюрихтеги Швейцария жогорку политехникалык мектебинин профессору Никлаус Вирт тарабынан студенттерди компиляторлорду түзүү, иштеп чыгуу усулдарына үйрөтүү ниетинде иштелип чыгылган.

Н. Вирттин ал учурда төмөндөгүдөй максаты болгон:

- көп эмес сандагы базалык түшүнүктөрдүн негизинде программа түзүүгө мүмкүн болгон;
- жөнөкөй синтаксиске ээ болгон;
- программаны машиналык кодго которуу жөнөкөй компилятордун жардамында ишке ашырууга мүмкүн боло тургандай программалоо тилин түзүү эле.

Андан бери Паскаль тилинин бир нече версиялары иштелип чыгылды. Андан сырткары кайсы бир жылдары Паскаль стандарты деген түшүнүк пайда болду. Бул версиялардын ар бири тилдин каражаттарына кандайдыр бир жаңылыктарды кошуу же тактоолорду, түзөтүүлөрдү кийрүү менен тилдин мүмкүнчүлүктөрүн кеңейтирип отурду. Анын ичинде талдоонун интегралданган чөйрөсүн түзүүгө болгон аракеттер болду. Акыбетинде Паскаль тилинин негизинде Turbo Pascal программалоо системасы пайда болду.

Паскаль программалоо тилинин артыкчылыктары:

- окуп үйрөнүүгө жөнөкөй, ачык айкын, программалоо жаатында биринчи үйрөнүлүүчү эң сонун тил болуп ал жакшы программалоо стилине үйрөтөт;
- структуралык программалоонун шарттарын канаатандырат: ал программалоонун башкаруучу структураларын реализациялоочу операторлорго ээ. Мындагы берилгендердин структурасынын жакшы тандалышы бул тилидин чөйрөсүндө жөнөкөй жана эффективдүү алгоритмдерди талдап чыгууга мүмкүнчүлүк берет;
- типтердин жана берилгендердин жакшы курамына ээ;
- тилдин конкреттүү ализациясы ПК нын бардык аппараттык кражаттарын пайдаланууга мүмкүнчүлүк берет:

- бул тилдин негизинде азыркы визуалдык программалоо системаларынын бири - Delphi иштелип чыккан.

Алибетте ар нерсенин эки жагы бар дегендей бул тилдин жетишпеген жагы жок деп айтууга болбойт:

- мында даражага көтөрүү амалы жок, бирок аны тилдин башка каражаттарынын жардамында ишке ашырууга болот;
- тилдин мүмкүнчүлүктөрүнүн көптүгүнөн улам бул тилди алгач ирет эле толук үйрөнүп чыгуу жөнөкөй иш эмес.

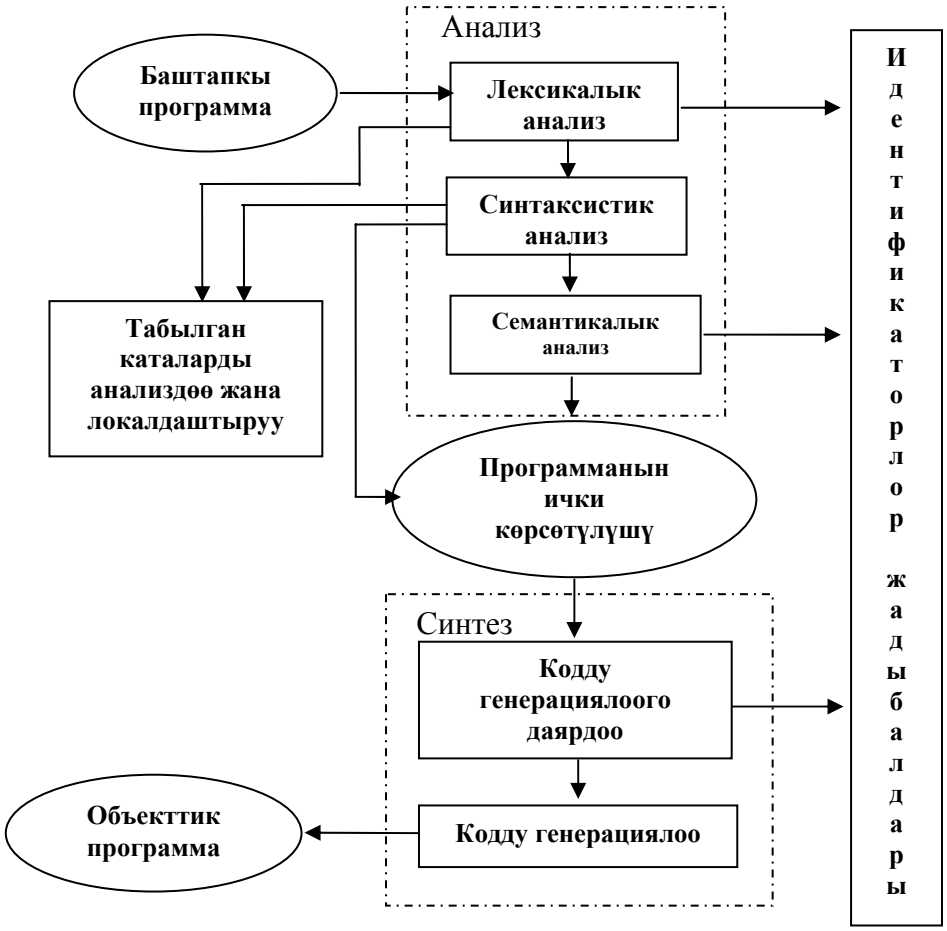
Бул китепте Turbo Pascal программалоо системасына кирген Borland Pascal 7.0 талдоонун интегралданган чөйрөсүндө программалоонун негиздери каралмакчы.

1. BORLAND PASCAL WITH OBJECTS 7.0 - ПРОГРАММАЛАРДЫ ИШТЕП ЧЫГУУНУН ПРОФЕССИОНАЛДЫК ПАКЕТИ

Эмгектене билүү – адам үчүн түгөнгүс байлык.
(Эзоп)

1.1. Компилятордун модели

Borland Pascal with Objects 7.0 программаларды иштеп чыгуунун профессионалдык пакетинин курамына кирген Turbo Pascal тили да жогорку деңгээлдеги программалоо тили болуп, мындай тилдерде түзүлгөн программаларды компьютер тарабынан аткарылуучу машиналык коддордун тилине которуу трансляторлор деп аталуучу атайын программалар аркылуу ишке ашырылат. Иштөө мүнөзү боюнча трансляторлор ***компиляторлор*** жана ***интерпретаторлор*** деп экиге бөлүнөт. Pascal тилинин трансляторлору анын эң алгачкы реализациясынан баштап акыркы Turbo Pascal 7.0 версиясына чейин компиляциялоочу принцип боюнча иштешет. Turbo Pascal тилинин базалык конструкцияларын бир кыйла тереңирээк түшүнүү үчүн төмөнкү сүрөттө көрсөтүлгөн компилятордун моделин карайбыз.



1.1 - сүрөт. Компилятордун модели.

Бул схемадан көрүнүп тургандай компиляция процесси жалпысынан негизги эки этаптан турат: **анализ** жана **синтез**.

Анализ этабында баштапкы программанын текстин тааныу (распознавание), идентификаторлор жадыбалдарын түзүү жана толтуруу сыяктуу иштер аткарылат. Анын ишинин натыйжасы болуп программанын компилятор үчүн түшүнүктүү болгон кайсы бир ички кодунун алынышы эсептелет.

Синтез этабында программанын ички көрсөтүлүшүнүн жана идентификаторлор жадыбалдарындагы информациялардын негизинде жыйынтык программанын тексти жаралат. Бул этаптын жыйынтыгы - объекттик коддун түзүлүшү.

Бул этаптар өз кезегинде компиляциянын фазалары деп аталган майда этаптардан турат. Биз жогоруда келтирген схемада компиляциянын фазалары жалпы көрүнүштө берилген. Алардан конкреттүү реализациясы жана бири-бири менен аракет этүү процесси, алибетте, компилятордун конкреттүү версиясынан көз каранды болот. Бирок, мында көрсөтүлгөн фазалар тигил же бул көрүнүштө иш жүзүндө дайыма ар бир конкреттүү компилятордо болот. Бул фазалардын ар бирине кыскача токтолуп өтөлү.

1.1.1. Лексикалык анализатор

Лексикалык анализатордун кирүүсүнө (вход) баштапкы программанын тексти келип түшөт, ал эми чыгуусундагы (выход) информация синтаксистик талдоо этабына берилет.

Жогорку деңгээлдеги тилде (ЖДТ) жазылган баштапкы программа ал программадагы бардык жолчолорду бири бирине удалаш туташтыруу менен түзүлгөн чынжырча болуп эсептелет. Тил үчүн колдонууга уруксат берилген символдордун ичинен дайыма символдор-ажыраткычтар деп аталуучу бир нече символдорду бөлүп көрсөтүүгө болот жана алар аркылуу баштапкы программанын сүйлөмдөрү айрым-айрым өзүнчө сөздөргө бөлүнөт. Мындай сөздөр компиляция теориясында ***лексемалар*** деп аталат.

Мисалы, төмөнкү сүйлөм (оператор)

```
if k: = 9 then writeln ( k );
```

мына мындай лексемаларга

```
if, k, :=, 9, then, writeln, (, k, ), ;
```

бөлүнөт.

Бул жерде ажыраткыч катары «пробел» символу пайдаланылат. Бирок, кээ бир символдордун арасында пробел турбагандыгын байкоого болот. Бул ал лексемалар өздөрү ажыраткычтар болуп

эсептелери менен түшүндүрүлөт, ошондуктан аларды башка лексемалардан ажыратып бөлүү үчүн атайын символ-ажыраткычтарды пайдалануу сөзсүз шарт эмес, бирок колдоно берүүгө болот. Мисалы, жогорку эле сүйлөмдү маанисин өзгөртпөстөн төмөнкүчө жазууга болот:

```
if k:= 9 then writeln(k);
```

Жалпы эреже:

Кандайдыр бир жерде бир символ-ажыраткыч турушу мүмкүн болсо, ал жерде каалагандай сандагы символ-ажыраткычтарды коюуга болот. Бирок бул эреже лексемалар-ажыраткычтар үчүн туура болбойт, анткени аларга кандайдыр бир маанидеги жүк (смысловая нагрузка) артылган.

1.1.2. Синтаксистик анализатор

Синтаксистик талдоо – анализ этабындагы негизги бөлүк болуп эсептелет. Ал лексикалык анализатор иштеп чыккан баштапкы программанын текстиндеги синтаксистик конструкцияларды бөлүп алууну аткарат. Компиляциянын ушул фазасында тилдин грамматикасынын сиснтаксистик эрежелеринин негизинде программанын сүйлөмдөрүнүн жазылышынын корректтүүлүгү (тууралыгы) текшерилет жана лексемалардын удаалаштыгын компилятордун ички коддорунун удаалаштыгына которуу жүргүзүлөт. Бул удаалаштык компьютер аткаруу керек болгон аракеттердин тартибин дээрлик чагылдырат, бирок акыркы машиналык код боло албайт. Ошентип синтаксистик талдоо башкы ролду - кирүүчү программалоо тилди тааануу (распознавание) ролун ойнойт.

Компиляция теориясында компилятордун ички коддорунун бир нече түрлөрү (триадалар, тетрадалар, ПОЛИЗ, даректар, атрибутташтырылган даректар, р-код) иштелип чыгылган, бирок аларды кароо биздин китептин рамкасынан сырткары жатат.

1.1.3. Семантикалык анализ

Семантикалык анализ – бул бөлүк кирүүчү тилдин семанткасынын көз карашында баштапкы программанын текстинин тууралыгын текшерет. Андан сырткары семантикалык анализ кирүү тилинин семантикасы талап кылуучу текстти өзгөртүп түзүүлөрдү (мисалы, типтерди айкын эмес өзгөртүп түзүү функцияларын кошуу) аткарышы керек.

1.1.4. Кодду генерациялоого даярдоо

Бул фаза жыйынтык программанын (бирок чыгуу тилиндеги тексттин пайда болушуна али алып барбаган) текстин синтездөө менен түздөн-түз байланышкан алдын ала аракеттерди аткарат. Адатта бул этапка тилдин элементтерин идентфикациялоо, эсти бөлүштүрүү жана ушул сыяктуу аракеттер кирет.

1.1.5. Кодду генерациялоо

Бул фаза чыгуу тилинин сүйлөмдөрүн түзүүчү командалардын жаралышы жана бүтүндөй жыйынтык программанын тексти менен байланышкан. Андан сырткары бул фаза адатта оптималдаштырууну - жаралган текстти (жыйынтык-программаны) кайрадан иштеп чыгуу процессин камтыйт. Оптималдаштыруу жыйынтык-программанын сапатына жана эффективдүүлүгүнө чоң таасир берет.

Ошентип код генератору компилятордун ички кодун компьютердин акыркы машиналык кодуна которууну ишке ашырат.

1.1.6. Идентификаторлор жадыбалдары

Идентификаторлор жадыбалдары (кээде символдор таблицасы деп да аташат) – бул атайын жол менен уюштурулган берилгендердин жыйындысы болуп алар баштапкы программанын элементтери жөнүндөгү информацияларды сактоо үчүн кызмат кылат.

Иштөө процессинде компилятордун жогорууда каралган бардык блоктору ушул жадыбалдарга кайрылат, анда бардык программаларды трансляциялоо үчүн туруктуу информация (мисалы, резервдештирилген сөздөрдүн таблицасы), ошондой эле ар бир программа үчүн жекече түрдө керек болгон (мисалы, идентификаторлордун, литералардын ж.б. таблицасы) информациялар жайгаштырылат. Ал информациялар кийин жыйынтык-программанын текстин түзүү үчүн пайдаланылат. Компилятордун конкреттүү реализациясында идентификаторлор жадыбалы бирөө же мындай жадыбал бир нече болушу мүмкүн.

Биз 1.1 - сүрөттө келтирген компиляция процессинин фазаларга бөлүнүшү усулдук максатты көздөйт, иш жүзүндө так мына ушундай көрүнүштө болбостугу мүмкүн.

1.1.7. Borland Pascal 7.0 чөйрөсүнүн структурасы

Borland фирмасынын Turbo Pascal тилинде программаларды талдап чыгуу үчүн иштеп чыккан мурдагы программалык

каражаттары копилятордон жана стандарттык процедуралардын жана функциялардын библиотекаларын өз ичине камтыган модулдардын жыйындысынан турган. Компилятор эки версияга ээ болуп алардын бирөө талдоонун интегралданган чөйрөсүндө (ИСР - интегрированная среды разработки – **ТИЧ** - ***талдоонун интегралданган чөйрөсү***- TURBO.EXE файлы) ал эми экинчиси пакетик режимде (TPC.EXE файлы) иштеген. Жаңы программалык каражат болгон Borland Pascal with Object7.0 компилятору – мурдагылардан бир топ айырмаланат. Анын структуралык схемасы 1.2-сүрөттө көрсөтүлгөн. Бул схеманын блокторунун арасындагы өз ара байланыштар жетишээрлик татаал экендигин байкоого болот. Бирок, иш жүзүндө бул өз ара байланыштар мындан да татаал. Мисалы Windows каптамасынын (оболочка) Program Manager блогунан андагы үч ТИЧ терининин ичинен каалаган бирөөсүн чакырууга болот. Схема ар түрдүү белгилөөлөр менен ашыкча татаалданып кетпеси үчүн реализациянын кээ бир детальдары каралган эмес.

Схемадан көрүнүп тургандай, Borland Pascal with Object7.0 компилятору үч ТИЧке (TURBO.EXE, BP.EXE, BPW.EXE) жана эки пакеттик версияга (TPC.EXE, BPC.EXE) ээ болуп, аларды төмөнкүчө тайпаларга ажыратууга болот:

MS-DOSтун башкаруусу алдында процессордун реалдык режиминде иштөөчү компилятордун версиялары (TURBO.EXE, TPC.EXE);

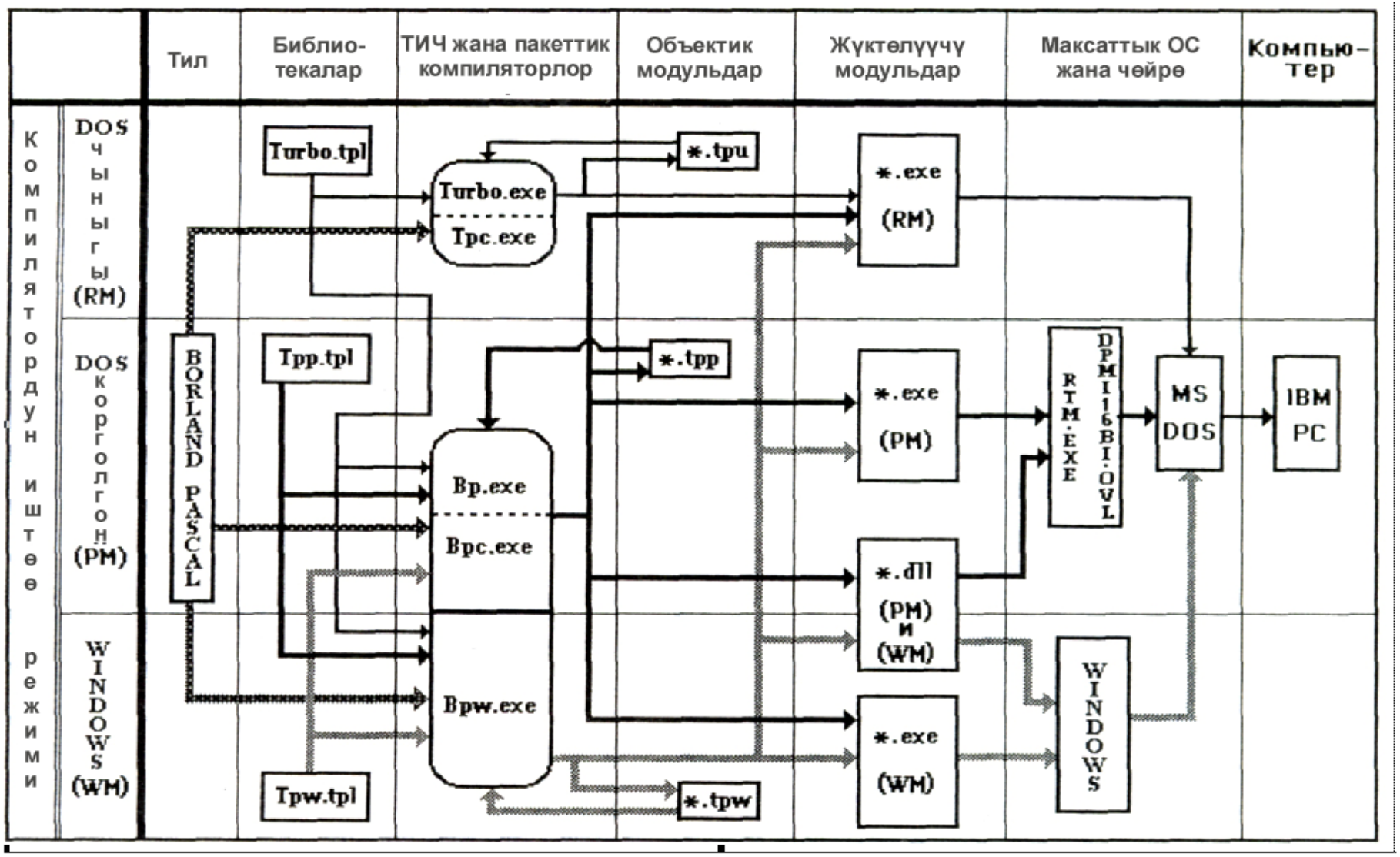
MS-DOSтун башкаруусу алдында процессордун корголгон режиминде иштөөчү компилятордун версиялары (BP.EXE жана BPC.EXE);

Windows дун башкаруусу алдында иштөөчү компилятордун версиясы (BPW.EXE).

Borland Pascal with Object7.0 компиляторунун версияларынын мүмүкүнчүлүктөрүн салыштыруу үчүн бирдей план боюнча алардын кээ бир мүнөздөмөлөрүн келтиребиз.

MS-DOS тун башкаруусу астында процессордун реалдык режиминде иштөөчү компилятордун версиялары (TURBO.EXE, TPC.EXE)

1. Компилятордун версиялары:
а) ТИЧте иштөөчү компилятордун версиясы (TURBO. EXE).
б) Компилятордун пакеттик версиясы (TPC. EXE).
2. Компилятордун жумушчу операциялык чөйрөсү: процессордун реалдык режиминдеги MS-DOS.



1.2 - сүрөт. Borland Pascal with Objects 7.0.
компиляторунун структуралык схемасы

3. Компилятордун чыгуу коду менен иштөөчү операциялык чөйрөлөр:

а) процессордун реалдык режиминдеги MS-DOS;

4. Чыгуу файлдарынын типтери: .EXE, .TPU.

5. Негизги модулдар – библиотекалар файлы: TURBO.TPL.

MS-DOS тун башкаруусу астында процессордун корголгон режиминде иштөөчү компилятордун версиялары (BP.EXE , BPC.EXE)

1. Компилятордун версиялары:

а) ТИЧте иштөөчү компилятордун версиясы (BP. EXE).

б) Компилятордун пакеттик версиясы (BPC. EXE).

2. Компилятордун жумушчу операциялык чөйрөсү: процессордун корголгон режиминдеги MS-DOS.

3. Компилятордун чыгуу коду менен иштөөчү операциялык чөйрөлөр:

а) процессордун реалдык режиминдеги MS-DOS;

б) процессордун корголгон режиминдеги MS-DOS;

в) Windows.

4. Чыгуу файлдарынын типтери: .EXE, .TPU, .TPP, .TPW, .DLL.

5. Негизги модулдар – библиотекалар файлы: .TPP, .TPL.

Windows дун башкаруусу астында иштөөчү компилятордун версиясы (BPW.EXE)

1. Компилятордун версиялары:

а) ТИЧте иштөөчү компилятордун версиясы (BPW.EXE);

б) Компилятордун пакеттик версиясы.

2. Компилятордун жумушчу операциялык чөйрөсү: Windows.

3. Компилятордун чыгуу коду менен иштөөчү операциялык чөйрөлөр:

а) процессордун реалдык режиминдеги MS-DOS;

б) процессордун корголгон режиминдеги MS-DOS;

в) Windows.

4. Чыгуу файлдарынын типтери: .EXE, .TPU, .TPP, .TPW, .DLL.

5. Негизги модулдар – библиотекалар файлы: TPW. TPL.

1.2. Borland Pascal программаларды талдоо чөйрөсү

Borland Pascal with Object 7.0 – программалоо системасынын курамына биз жогоруда байкагандай үч программалоо чөйрөсү кирет: Borland Pascal for Windows (BPW), Borland Pascal (BP) жана Турбо Паскаль (TP). BPW чөйрөсү Windows операциялык системасынын башкаруусу астында иштөөчү программаларды түзүү үчүн, Турбо Паскаль (TP) чөйрөсү MS DOS менен иштөө үчүн арналган. Ал эми BP чөйрөсү MS DOS үчүн да, Windows үчүн да программаларды түзүүгө мүмкүнчүлүк берүү менен жогорку эки системанын ортосундагы өзгөчө бир көпүрө болуп эсептелет. Турбо Паскаль чөйрөсүндө түзүлгөн баардык программалар эч кандай өзгөртүүсүз BP чөйрөсүндө трансляцияланышы мүмкүн. Бирок тескерисинче эмес, анткени BP бир кыйла кубаттуу мүмкүнчүлүктөргө ээ.

BP чөйрөсүндө программаларды түзүүнү көздөгөн пайдалануучулар үчүн бул үч чөйрөнүн ортосундагы айырмачылыктарды жана артыкчылыктарды билүү бир топ пайдалуу.

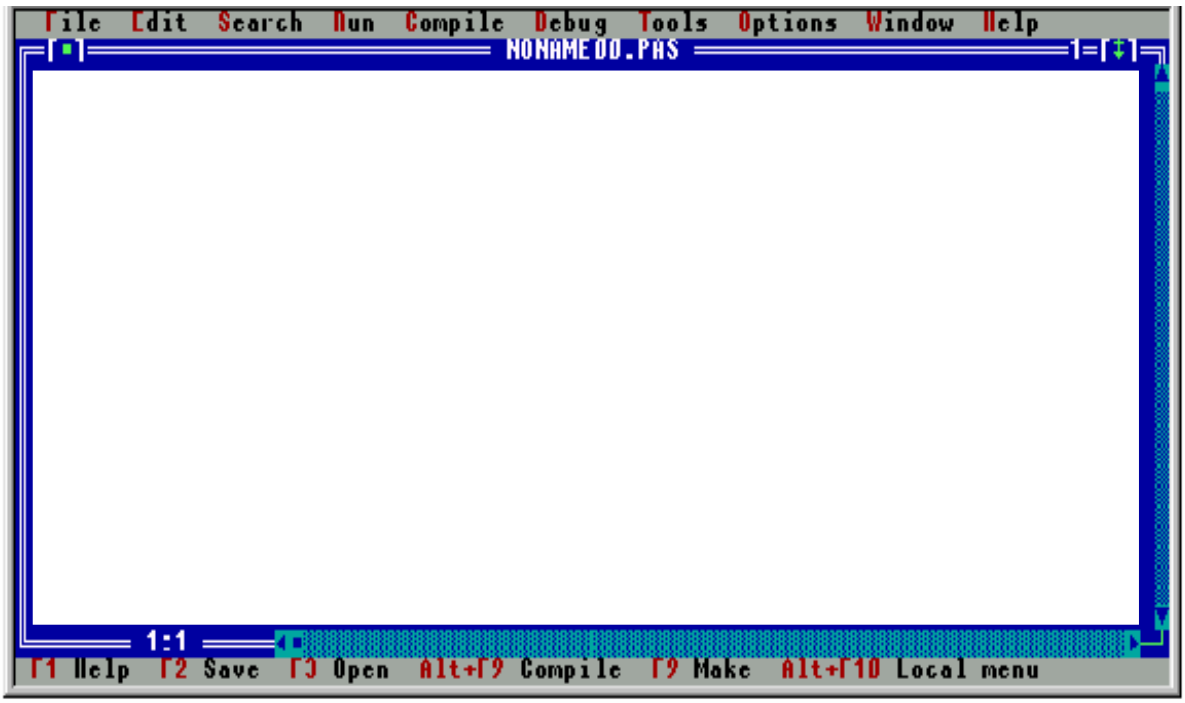
1.2.1. BP – талдоонун интегралданган чөйрөсүн ишке салуу

BP – талдоонун интегралданган чөйрөсү BP.EXE файлында кармалып турат жана ал MS DOS тун башкаруу астында иштөөчү каалаган башка программа сыяктуу эле, мисалы,

```
\BP\ bin\ bp
```

командасы аркылуу (BP.EXE файлы BP каталогунун BIN камтылуучу каталогунда жайгашкан стандарттык жайлашуу учурунда) ишке салынышы мүмкүн. Ал эми Windows операциялык системасынын чөйрөсүндө ар кандай эле DOS-программа катары ишке салынат. Ишке салгандан кийин ал дароо эле экранда TURBO.EXE чөйрөсүнүн терезесинен дээрлик айырмаланбаган терезени ачат. Бул терезе жайгашкан экран интегралданган чөйрөнүн негизги экраны деп аталат, ал төмөнкүдөй ар түрдүү функционалдык арналышка ээ болгон үч бөлүктөн турат:

- менюлар жолчосу;
- жумушчу зона;
- абалдар жолчосу.



BP чөйрөсүнүн пайдалануучулук интерфейсинин мүмкүнчүлүктөрүн тез өздөштүрүү үчүн оболу ушул бөлүктөрдү өздөштүрүү зарыл.

1.2.2. BP чөйрөсүнүн менюсу

Интегралданган чөйрөнүн менюсун башкы меню жана камтылма менюлар деп экиге бөлүп карооо болот.

BP.EXE файлы активдештирилгенден кийин курсор негизги экрандын жумушчу зонасында жайланышкан болот. Чөйрөнүн башкы менюсуна кирүү **F10** клавишасын басуу менен аткарылат.

Башкы меню он камтылма менюдан туруп төмөнкүдөй көрүнүшкө ээ:

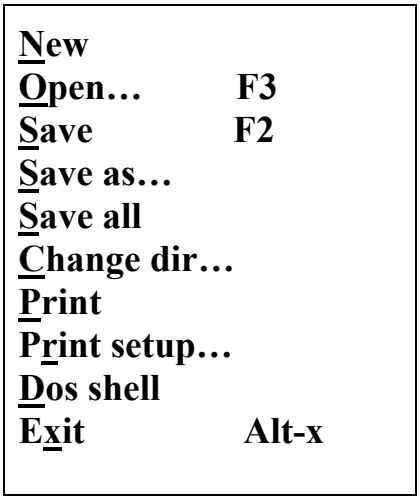
File Edit Search Run Compile Debug Tools Options Window Help

File менюсу - файлдар менен иштөө үчүн командаларды камтыйт.
Edit менюсу - программанын текстин редактирлөө командаларын камтыйт.
Search менюсу - программанын компоненттерин тез издөө жана программанын фрагменттерин контекстик алмаштыруу командаларын камтыйт.
Run менюсу - программанын аткарылышын ишке салуу жана анын ишин трассирлөө командаларын камтыйт.
Compile менюсу - компиляция үчүн ар түрдүү режимдерди берүү командаларын камтыйт.

Debug менюсу - оңдоп-түзөө (отладка) информацияларын башкаруу командаларын камтыйт.
Tools менюсу - интегралданган чөйрөнүн билдирүүлөрү менен иштөө командаларын жана оңдоп-түздөө утилиталарын чакыруу командаларын камтыйт.
Options менюсу - ТИЧтин опцияларын коюу үчүн диалогдук терезени алуу командаларын камтыйт.
Window менюсу -ТИЧтин терезелери менен иштөө командаларын камтыйт.
Help менюсу - талдоонун интегралдык чөйрөсүндө иштөө жана Turbo Pascal боюнча сурап билүү информацияларын чакыруу командаларын камтыйт.

File менюсу

File менюсу он командадан туруп төмөнкүдөй көрүнүшкө ээ:



New - редактирлөөнүн (Edit) жаңы терезесин ачуу жана аны жаңы программаны кийрүү үчүн активдештирүү.
Open…- дискте редактирлөө жана ишке салуу үчүн зарыл болгон файлды тандоо үчүн **Open a File** - диалогдук терезесин чакыруу.
Save - редактирлөөнүн активдүү терезесиндеги программаны дисктеги файлга эски аты менен сактоо.
Save as… - редактирлөөнүн активдүү терезесиндеги программаны сактоо үчүн файлга жаңы ат көрсөтүүгө мүмкүнчүлүк берүүчү диалогдук терезесин чакыруу.
Save all - редактирлөөнүн терезелериндеги бардык файлдарды сактоо.
Change dir… - учурдагы каталогдун атын коюу (установка).
Print - редактирлөөнүн активдүү терезесиндеги программаны печаттоо.

Print setup… - печаттоонун параметрлерин коюу үчүн Print Setup диалогдук терезесин чакыруу.
Dos Shell - Exit командасы менен кайра ТИЧке кайра кайтуу мүмкүнчүлүгү менен DOSко убактылуу чыгуу.
Exit - ТИЧтен чыгуу (ишти аяктоо).

Edit менюсу

Edit менюсу жети командадан туруп төмөнк көрүнүшкө ээ.

Undo	**Ait+BkSp**
Redo	
Cut	**Shift+Del**
Copy	**Ctrl+Ins**
Paste	**Shift+Ins**
Clear	**Ctrl+Del**
Show clipboard	

Undo - редактирлөөдөгү акыркы аракетти алып салуу.
Redo - редактирлөөдөгү акыркы аракетти кайталоо.
Cut - программадагы бөлүнүп коюлган фрагментти жоготот жана аны Clipboard - буферине жайгаштырат.
Copy - программанын бөлүнүп коюлган фрагментин Clipboard буферине көчүрөт.
Show clipboard – clipboard буферинин терезесин экранга чыгарат.

Search менюсу

Search менюсу он эки командадан туруп көрүнүшү төмөнкүдөй:

Find…
Replace…
Search again
Go to line number
Show last compiler error
Find error…
Find Procedure…
Previous browser
Objects
Units
Globals
Symbol…

Find… - программанын текстинде Find диалогдун терезесинде көрсөтүлгөн фрагменттин жайгашкан ордун табуу.

Replace… - программанын текстинде Replace диалогдук терезесинде көрсөтүлгөн фрагменттин жайгашкан ордун табуу жана аны ушул эле диалогдун терезесинде көрсөтүлгөн жаны фрагмент менен алмаштыруу.
Search again - мурдагы Find - издөө же Replace - алмаштыруу командасын кайталайт.
Go to line number… - программанын номери диалогдук терезеде көрсөтүлгөн жолчосуна өтүү.
Show last compiler error - компиляциянын мурдагы катасы жөнүндөгү билдирүүнү экрандын төмөнкү жолчосуна чыгарат жана курсорду ката табылган жерге коет.
Find error… - программанын текстинде ачылган диалогдук терезеде берилген адрес боюнча аткаруу убактысынын катасы жайгашкан жерди табуу.
Find procedure… - программанын текстинде аты ачылган диалогдук терезеде көрсөтүлгөн процедураны издөө. Бул команданы кадамдап ондоп-түзөө (пошаговая отладка) режиминде гана колдонууга болот.
Previons browser - мурдагы browse терезесиндеги информасияны чакыруу.
Objects - Browse терезесинде учурдагы программанын бардык обьекттеринин иерархиясын дарак көрүнүшүндө көрсөтөт.
Units – Browse терезесинде учурдагы программанын модулдары жөнүндөгү информацияны көрсөтөт.
Globals - Browse терезесинде учурдагы программанын бардык глобалдык өзгөрүлмөлөрүн көрсөтөт.
Symbol… - учурдагы программанын пайдаланууучу жолчолордун номерлеринин жыйындысы берилүүчү идентификаторун берүүгө мүмкүнчүлүк берет.

Run менюсу

Run менюсу алты командадан туруп көрүнүшү төмөнкүдөй:

Run	**Ctrl+ F9**
Step over	**F8**
Trace into	**F7**
Goto cursor	**F4**
Program reset	**Ctrl+F2**
Parameters…	

Run - программаны аткаруу үчүн ишке салуу (запуск). Ишке салууда талап кылынган параметрлер ушул эле менюда көрсөтүлгөн Parameters… командасынын жардамында көрсөтүлөт.

Step over - программаны кадамдап аткаруу. Процедураны жана функцияны чакыруу бир оператор (бир кадам) катары аткарылат.
Trace into - программаны кадамдап аткаруу. Процедураны же функцияны чакырууда анын текстине кирүү жана андагы операторлорду кадамдап аткаруу болуп өтөт.
Go to cursor – программанын кадамдап аткаруунун учурдагы жолчосунан баштап курсор турган жолчосуна чейинки бөлүгүн аткаруу.
Program reset - программаны оңдоп-түзөө (отладка) сеансын аяктайт жана ал ээлеп турган эсти бошотот.
Parameters… - аткарылуучу программа үчүн параметрлер көрсөтүлүүчү диалогдук терезени ачат.

Compile менюсу

Compile менюсу жети командадан туруп төмөндөгүдөй көрүнүшкө ээ.

Compile	**Alt+F9**
Make	**F9**
Build	
Target…	**Real**
Primary file…	
Clear primary	
Information…	

Compile - редактирлөөнүн активдүү терезесинде турган файлды компиляциялоо.
Make - көп модулдуу программаны .EXE файлды түзүү менен шарттуу компиляциялоо. Эгерде акыркы компиляциядан бери кээ бир модулдарга өзгөртүүлөр киргизилген болсо, анда Make командасы аткарылганда өзгөргөн жана андан көз каранды болгон модулдар гана кайра компиляцияланат.
Build - .EXE файлды түзүү менен көп модулдуу программаны шарттуу компиляциялоо. Акыркы компиляциядан кийин өзгөртүү киргизилдиби, жокпу андан көз карандысыз программанын бардык модулдары кайрадан компляцияланат.
Target… - Target диалогдук терезесинде тиркеме (приложение) үчүн колдонулуучу (целевая) платформаны тандоо. Мүмкүн болгон варианттар: Real mode Application (реалдык режимдин тиркемеси), Protected mode Application (корголгон режимдик тиркемеси), Windows Application (Windows -тиркеме).

Primary file… - Make жана Build командаларын аткаруу үчүн компиляцияланууучу программанын башкы (главный) файлын көрсөтүү үчүн диалогдук терезени ачат.
Clear primary file - башкы компиляцияланууучу файлды көрсөтүүнү алып салуу.
Information… - компиляцияланган файл жөнүндөгү информацияны кармап турган диалогдук терезени ачат.

Debug менюсу

Debug менюсу тогуз командадан туруп төмөнкүдөй көрүнүшкө ээ:

Breakpoints	
Call stack	**Ctrl+F3**
Register	
Watch	
Output	
User screen	**Alt+F5**
Evaluate / modify…	**Ctrl+F4**
Add watch …	**Ctrl+F7**
Add breakpoint	

Breakpoints - Breakpoints диалогдук терезесин ачып, андагы командалардын жардамында үзгүлтүктүн шартсыз чекиттерин башкаруу мүмкүн, б.а.оңдоп-түзөө (отладочные) иштерин аткаруу үчүн токтотулуп турулган программанын текстиндеги чекиттерди башкаруу үчүн.
Call stack - Call Stack терезесин ачат, ал жерде берилген моментте аткарылып жаткан процедурага чейинки чакырылган программанын процедураларынын аттарынын удаалаштыгы көрсөтүлөт.
Register - процессордун регистрлери жөнүндөгү информацияны кармап турган Register терезесин ачат.
Watch - Watch терезесин ачат, бул терезеде өзү үчүн пайдаланууучу оңдоп-түзөө кезинде талап кылынган программанын өзгөрүлмөлөрүнүн жана туюнтмаларынын маанилери жөнүндөгү информацияларды чыгара алат.
Output - Output терезесин ачат, ал терезеде DOS экраны жана программанын ишинин жыйынтыктары (графиканы айтпаганда) чагылтылат.
User screen - толук экран режиминде графиканы да кошкондо, программанын шинин жыйынтыктарын көрүү.

Evaluate/modify… - Evaluate and modify терезесин ачат, мында маанисин аныктоо талап кылынган туюнтманы көрсөтүүгө, программадагы өзгөрүлмөлөрдүн жана берилгендердин элементтеринин маанилерин карап көрүүгө жана аларды өзгөртүүгө болот.
Add watch… - Add watch диалогдук терезесин ачат, анда программист маанилери оңдоп-түзөөнү (отладканы) аткарууда кызыктырган туюнтманы же өзгөрүлмөнүн атын көрсөтө алат.
Add breakpoint… - үзгүлтүктөрдүн шарттуу жана шартсыз чекиттери коюлуучу Add breakpoint диалогдук терезесин ачат.

Tools менюсу

Tools менюсу жети командадан туруп көрүнүшү төмөнкүдөй:

Messages	
Go to next	**Alt+F8**
Go to previous	**Alt+F7**
Grep	**Shift+F2**
Turbo Assembler	**Shift+F3**
Turbo Debugger	**Shift+F4**
Turbo Profiler	**Shift+F5**

Messages - DOStун фильтри аркылуу чыгарууну аткарып жаткан программалардын билдирүүлөрүү чагылдырылуучу Messages терезесин ачат. (Мисалы, GREP программасынын билдирүүлөрү) Ондолуп-төзөлүп (отлаживаемая) жаткан программанын билдирүүгө тиешелеш келген жолчосуна өтүү үчүн, курсорду ушул билдирүүгө коюп кийин ENTER клавишасын басуу керек.
Goto next - Messages же Browser терезесинин кийинки элементине өтүүнү аткарат.
Go to previons - Messages же Browser терезесинин мурдагы элементине отууну аткарат.
Grep - Grep программасын чакыруу.
Turbo Assembler - Turbo Assembler программасын чакыруу.
Turbo Debugger - Turbo Debugger программасын чакыруу.
Turbo Profiler - Turbo Profiler программасын чакыруу.

Options менюсу

Options менюсу он бир командадан туруп төмөнкүдөй көрүнүшкө ээ:

Compiler… - Compiler Options терезесин ачат, бул терезеде программист чыгуу кодун генерациялоону, аткарылуу этабында каталарды табууну жана ондоп-түзөө информацияларынын жакшы иликтениш (детально) денгээлин башкаруу үчүн опцияларды тандай алат.

Memory sizes… - Memory sizes диалогдук терезесин ачат, мында программист программа үчүн стек жана динамикалык өзгөрүлмөлөр үчүн ажыратылуучу оперативдик эстин өлчөмүн бере алат.

Linker… - Linker диалогдук терезесин ачат, анда байланыштар редакторунун (редактор связей) ишин башкаруу мүмкүнчүлүктөрү коюлат.

Debugger… - Debugger диалогдук терезеси ачылат, мында интегралданган оңдоп-түзөөчүнүн (отладчиктин) ишин башкаруу мүмкүнчүлүктөрү коюлат.

Browser… - Browser Options диалогдук терезесин ачат, мында программист браузердин ишин башкаруу үчүн ар түрдүү опцияларды кое алат.

Tools… - Tools диалогдун терезесин ачат, мында программист Tools менюсуна программаларды ишке салуу командаларын кошууну же жоготууну, ошондой эле бул программаларды тескөөнү (настройка) аткара алат.

Environment ► - интегралданган чөйрөнүн сырткы көрүнүшүн жана анын көрсөтүлбөгөн учурда (по умалчанию) кабыл алынган опцияларын башкарууга боло турган алты командадан турган менюну кармаган терезени ачат.

Open… - Open Options диалогдук терезесин ачат, мында пайдалануучу TP кеңейтирилишине ээ болгон файлда Options менюсундагы Save командасы аркылуу сакталган интегралданган чөйрөнүн орнотулуштарын (установка) калыбына келтире алат.

Save - Search диалогдук терезесинде Compile менюсунун Primapy File командасы аркылуу жасалган опциялардын орнотулуштарын, ошондой эле, Options менюсунда аткарылган орнотулуштарды файлга сактайт.
Save as… - Save Options терезесин ачып, анда ТИЧтин учурдагы орнотулуштары (установка) сакталуучу каталогдун жана файлдын аттары көрсөтүлөт.

Window менюсу

Window менюсу он командадан туруп көрүнүшү төмөнкүдөй:

Tile	
Cascade	
Close all	
Refresh display	
Size/Move	**Ctrl+F5**
Zoom	**F5**
Next	**F6**
Previous	**Shift+F6**
Close	**Alt +F3**
List…	**Alt +0**

Tile - экранда редактирлөөнүн бардык ачылган терезелерин аларга экрандын барабар бөлүктөрүн бөлүү менен жайгаштырат.
Cascade – экранда редактирлөөнүн бардык ачылган теезелерин каскад көрүнүшүндө жайгаштырат.
Close all - бардык ачылган терезелерди жабуу.
Refresh display - информацияны экранга чакырууда алдын ала көрүүгө болбогон таштоо болгон учурда экранда ТИЧтин сүрөттөлүшүн калыбына келтирет.
Size/Move - экранда активдүү терезенин өлчөмүн же/жана анын позициясын өзгөртүү. Экран боюнча терезени жылдыруу жебече-клавишалар аркылуу аткарылат, ал эми анын өлчөмдөрүн өзгөртүү ушул эле клавишаларды **shift** менен бирге басуу аркылуу аткарылат.
Zoom - активдүү терезени толук экран өлчөмүнө чейин чоңойтот жана эгер ушундай өлчөмдө болгон болсо аны баштапкы абалына келтирет.
Next - Next командасын бир нече жолу удаалаш колдонуу, ачылган терезелердин циклдик түрдө алмашып активдешүүсүнө алып келет.
Previous - Next командасына окшош иштейт, бирок терезелерди тескери удаалаштыкта алмаштырып турат.
Close – активдүү терезени жабуу.

List… - Window List диалогдук терезесин ачат, мында ТИЧти ишке салгандан (запуск) баштап ачылган бардык терезелерди санап көрсөтөт.

Help менюсу

Help менюсу он үч командадан туруп көрүнүшү төмөнкүдөй:

Contents	
Index	**Shift+F1**
Topic seach	**Ctrl+F1**
Previous topic	**Alt+F1**
Using help	
Files…	
Compiler directives	
Procedures and functions	
Reserved words	
Standart units	
Borland Pascal Language	
Error messages	
About…	

Contents - сурап билүү системасынын мазмунун кармап турган Pascal Help Contents терезесин экранга чыгарат.
Index - сурап билүү системасында бар болгон терминдердин алфавиттик жыйындысын кармап турган Index терезесин экранга чыгарат.
Topic seach - активдүү терезеде курсор коюлган термин жөнүндөгү информацияны экранга чыгарат.
Previons topic - Help менюсунун мурдагы терезесине кайтуу. Бул команданын көп жолу аткарылышы Help менюсунун 20 га чейин удаалаш терезелерин тескери тартипте чыгарат.
Using Help - сурап билүү системасын колдонуу боюнча инструкцияны экранга чыгарат.
Files… - бул команданын опциялары жаңы Install Help Files терезесине кирүүнү камсыз кылат, мында сурап билүү системасына жаңы информациялык файлдарды кошууга жана пайдаланылбаган инфармацияларды жоготууга болот.
Compiler directives - экранга компилятордун бардык директивалары-нын тизмесин чыгарат.
Procedures and functions – экранга Turbo Pascal тилинин процедура-ларынын жана функцияларын алфавит боюнча издөө үчүн терезени чыгарат.

Reserved words - экранга Turbo Pascal тилинин резервделген сөздөрүнүн тизмесин чыгарат.
Standart units - Turbo Pascal тилинин стандарттык модулдарынын тизмесин экранга чыгарат.
Borland Pascal Language - Borland (Turbo) Pascal тилинин негизги түшүнүктөрүнүн жыйындысын экранга чыгарат.
Error messages - программада каталарды тапканда ТИЧ тарабынан берилүүчү билдирүүлөр жөнүндөгү информацияны кармап туруучу терезени экранга чыгарат.
About… - Borland Pascal With Objects пакетин иштеп чыккандар, версиялары жана автордук укуктар жөнүндө информацияны кармаган терезени экранга чыгарат.

1.2.3. Талдоонун интегралданган чөйрөсүнүн «ысык (горячие)» клавишалары

Жалпы арналыштагы клавишалар

F10 - ТИЧтин башкы менюсуна кирүү.
Esc - диалогдук терезени же меню терезесин жабуу.
Alt+X - ТИЧтен чыгуу.
Ctrl+Break-ишке салынган программанын аткарылыщын үзгүлтүккө учуратуу жана ТИЧке кайтуу. Ишке салынган программа циклдешип (зацикливание) же илинип (зависание) калганда пайдаланылат.
Print Screen - экрандын көчүрмөсүн принтерде печаттоо.
Pause - экрандагы өзгөрүп жаткан сүрөттөлүштү каалаган клавишаны басканга чейин токтотуп туруу.

Сурап билүү системасы менен иштөө клавишалары

F1 - ТИЧтин берилген моменттеги активдүү терезеси жөнүндөгү же курсор көрсөтүп турган менюнун командасы жөнүндөгү сурап билүү системасынын информациясын экранга чыгарат.
F1(эки жолу) - сурап билүү системасынан пайдалануу боюнча көрсөтмөлөрдү экранга чыгарат .
Ctrl+F1 - активдүү терезедеги курсор көрсөтүп турган термин жөнүндөгү информацияны экранга чыгарат.
Alt+F1 - Help- тин мурдагы терезесине кайтуу. Бул команданын көп жолку аткарылышы Help-тин 20-га чейинки терезелерин тескери тартипте чыгарат.
Shift+F1- сурап билүү системасынын терминдердин алфавиттик жыйындысын кармап турган Index терезесин экранга чыгарат.

Файлдарды ачуу, сактоо жана редактирлөөнүн терезелери менен иштөө клавишалары

F2 - редактирлөөнүн активдүү терезесиндеги программаны дискте эски ат менен файлга сактоо.

F3 - редактирлөө жана ишке салуу үчүн ачыш зарыл болгон файлды дискте тандоо үчүн Open a File диалогдук терезесин чакыруу.

Alt+F3 - редактирлөөнүн активдүү терезесин жабуу.

F6 - бул клавишанын бир нече жолу удаалаш басылышы ачылган терезелердин циклдик түрдө алмашышына алып келет.

Shift+F6 – бул F6 клавишасына окшош иштейт, бирок терезелерди тескери удаалаштыкта алмаштырат.

Alt+O – ТИЧ ишке салынган моменттен берки ачылган бардык терезелер саналып көрсөтүлгөн Window List диалогдук терезесин ачат.

F5 – активдүү терезени толук экран өлчөмүнө чейин кеңейтет жана эгер терезе мурда ушундай өлчөмдө турган болсо, анда аны кайра баштапкы абалга алып келет.

Ctrl+F5 – активдүү терезенин өлчөмүн жана/же анын экрандагы позициясын өзгөртүү. Терезени экран боюнча жылдыруу клавиша-жебечелер (← ↑ → ↓) жардамында, ал эми анын өлчөмдөрүн өзгөртүү - Shift жана клавиша-жебечелерди бир убакта басуу менен аткарылат.

Программанын текстинин фрагменттери менен иштөө клавишалары

Shift+"клавишалар-жебечелер"- программанын фрагментин бөлүп алуу.

Shift+ Del – программанын бөлүнүп алынган фрагментин өчүрүү жана аны Clipboard буферине жайгаштыруу.

Ctrl+Ins - программанын бөлүнүп алынган фрагментин Clipboard буферине көчүрүү.

Shift+Ins - Clipboard буферинде турган бөлүнүп алынган фрагментти активдүү терезенин курсор жайгашкан позициясына коюу.

Ctrl+Del - программанын бөлүнүп алынган фрагментин Clipboard буферине жайгаштырбастан өчүрүү.

Alt+ BakcSpace – редактирлөөдөгү акыркы аракетти жокко чыгаруу.

Компиляция жана аткаруу үчүн пайдаланылуучу клавишалар

Alt+F9 – редактирлөөнүн активдүү терезесинде турган файлды компиляциялоо.

F9 – көп модулдуу программаны .EXE файлды түзүү менен шарттуу компиляциялоо. Эгерде акыркы компиляциялоодон кийин кээ бир модулдарга өзгөртүүлөр киргизилген болсо, анда өзгөртүүлөр киргизилген модулдар жана ушул модулдардан көз каранда болгон модулдар гана кайра компиляцияланат. Жөнөкөй программалар үчүн Alt+F9 комбинациясына эквиваленттүү.

Ctrl+F9 – редактирлөөнүн активдүү терезесинде турган программаны аткаруу.

Программаларды оңдоп-түзөө (отладка) клавишалары

Alt+F5 – программанын аткарылышынын жайантыгын көрүү.

F8 – программаны кадамдап аткаруу. Процедураларды жана функцияларды чакыруу бир оператор(бир кадам) катары аткарылат.

F7 – программаны кадамдап аткаруу. Процедураларды жана функцияларды чакырууда анын текстинин ичине кирүү жана анан операорлорун кадамдап аткаруу болуп өтөт.

F4 – программанын кадамдап аткаруунун учурдагы жолчосунан баштап курсор турган жолчосуна чейинки бөлүгүн аткаруу.

Ctrl+F2 – программаны оңдоп түзөө сеансын аяктайт жана ал ээлеген эстин бөлүгүн бошотот.

Ctrl+F3 – берилген моментте аткарылып жаткан процедурага чейин чакырылган программанын процедураларынын аттарынын удаалаштыгы көрсөтүлгөн Call Stack терезесин ачат.

Ctrl+F4 – Evaluate and modify терезесин ачат. Мында маанисин аныктоо талап кылынган туюнтманы көрсөтүүгө жана программадагы өзгөрүлмөлөр менен берилгендер элементтеринин маанилерин көрүүгө жана аларды өзгөртүүгө болот.

Ctrl+F7 – программист оңдоп түзөө кезинде маанилери кызыктырган туюнтманын же өзгөрүлмөнүн атын көрсөтө ала турган Add Watch диалогдук терезесин ачат.

Талдоонун интегралданган чөйрөсүнүн (ТИЧтин) редактору

Курсорду жылдыруу командалары

Клавишалар	Аракети
←	курсорду бир символго солго жылдырат
→	курсорду бир символго оңго жылдырат
↑	курсорду бир символго жогору жылдырат
↓	курсорду бир символго төмөн жылдырат
Ctrl+←	курсорду бир сөзгө солго жылдырат

Клавишалар	Аракети
Ctrl+→	курсорду бир сөзгө оңго жылдырат
Ctrl+W	текстти бир жолчого жогору түрөт
Ctrl+Z	текстти бир жолчого төмөн түрөт
PgUp	текстти бир бетке жогору барактайт
PgDn	текстти бир бетке төмөн барактайт
Home	курсорду жолчонун башына жылдырат
End	курсорду жолчонун аягына жылдырат
Ctrl+Home	курсорду терезенин жогорку чегине жылдырат
Ctrl+End	курсорду терезенин төмөнкү чегине жылдырат
Ctrl+PgUp	курсорду файлдын башталышына жылдырат
Ctrl+PgDn	курсорду файлдын аягына жылдырат

Сыйлыгыштыруу жана өчүрүү командалары

Клавишалар	Аракети
Del	курсордун үстүндө турган символду өчүрүү
Backspasce	курсордун сол жагында турган символду өчүрүү
Ctrl+Y	курсор турган символду өчүрүү
Ctrl+QY	курсордон баштап жолчонун акырына чейин өчүрүү
Ctrl+T	курсордун оң жагындагы сөздү өчүрүү
Ctrl+N	курсордун позициясына жолчону сыйлыгыштыруу
Ins	символдорду сыйлыгыштыруу режимин коюу/алып салуу

Блоктор менен иштөөнүн стандарттык командалары

Клавишалар	Аракети
Ctrl+K B	учурдагы позицияны блоктун башталышы катары белгилөө
Ctrl+K K	учурдагы позицияны блоктун бүтүшү катары белгилөө
Ctrl+K L	учурдагы жолчону блок катары белгилөө
Ctrl+K T	учурдагы сөздү блок катары белгилөө
Ctrl+Q B	курсорду блоктун башталышына жылдырат
Ctrl+Q K	курсорду блоктун бүтүшүнө жылдырат
Ctrl+K H	белгиленген блокту бекитүү/көрсөтүү
Ctrl+K Y	блокту өчүрүү
Ctrl+K C	блокту көчүрүү
Ctrl+K V	блокту башка жерге которуу
Ctrl+K R	блокту дисктен окуу
Ctrl+K W	блокту дискке жазуу
Ctrl+K P	блокту печаттоо
Ctrl+K I	тексттин блок жайгашкан жолчолорун бир позицияга оңго жылдыруу
Ctrl+K U	тексттин блок жайгашкан жолчолорун бир позицияга солго жылдыруу

Блоктор менен иштөөнүн кошумча командалары

Клавишалар	Аракети
shift+←	блокту бир символ солго кеңейтет
shift+→	блокту бир символ оңго кеңейтет
shift+↑	блокту бир жолчого жогору кеңейтет
shift+↓	блокту бир жолчого төмөн кеңейтет
shift+End	блокту жолчонун аягына чейин кеңейтет
shift+Home	блокту жолчонун башталышына чейин кеңейтет
shift+PgDn	блокту бир бетке төмөн кеңейтет
shift+PgUp	блокту бир бетке жогору кеңейтет
shift+Crtl+←	блокту бир сөзгө солго кеңейтет
shift+Crtl+→	блокту бир сөзгө оңго кеңейтет
shift+Crtl+End	блокту файлдын аягына чейин кеңейтет
shift+Crtl+Home	блокту файлдын башталышына чейин кеңейтет
Crtl+Ins	белгиленген блокту Clipboard буферине көчүрүү
shift+Del	белгиленген блокту аны баштапкы текстен өчүрүү менен Clipboard буферине көчүрүү
Ctrl+Del	баштапкы тексттен белгиленген блокту Clipboard буферине жайгаштырбастан өчүрүү
Shift+Ins	Clipboard буферинен андагы белгиленген блокту редактирлөө терезесине курсор турган позицияга коюу

Редактирлөөнүн дагы башка командалары

Клавишалар	Аракети
Crtl+Q F	Find издөө терезесин ачуу
Crtl+L	акыркы издөөнү кайталоо
Crtl+Q[же Crtl+Q]	курсор көрсөтүп турган кашаага жуп болгон кашааны табуу
Crtl+Q A	Replace контексттик алмаштыруу терезесин ачуу
Crtl+P	башкаруу символун коюу
Alt+Backspace	(Undo) өзгөрүүлөрдү алып салуу
Crtl+O I	Автоматтык таштоо (отступ)режимин коюу/алып салуу
Crtl+O U	Backspase клавишасы үчүн структуралык (отступтар) режимин коюу/алып салуу
Crtl+O R	табуляциянын позициялары боюнча курсорду жылдыруу режимин коюу/алып салуу
Crtl+O F	оптималдык толтуруу режимин коюу/алып салуу
Crtl+O T	табуляцияны пайдалануу режимин коюу/алып салуу

Жаңы үйрөнчүктөр үчүн кээ бир маалыматтар

Башкы менюга кирүү үчүн - **F10** клавишасын басуу керек.

Аракеттерди аткарбастан менюдан же диалогдук терезеден чыгуу үчүн – Esc клавишасын басуу керек.

«Ысык клавишаларды» басуу эрежелери

Turbo Pascal тилинде «ысык клавишалардын» үч түрү колдонулат:

1. Төмөнкүчө жазылуучу эки клавишанын комбинациясы

Биринчи_клавиша + Экинчи_клавиша (мисалы, Alt+F)

төмөнкүчө басылат: адегенде Биринчи_клавишаны(Alt) басып, аны кое бербестен туруп Экинчи_клавишаны(F) басуу керек.

2. Төмөнкүчө жазылуучу үч клавишанын комбинациясы

Биринчи_клавиша + Экинчи_клавиша + Үчүнчү_клавиша (мисалы, Ctrl+K B)

төмөнкүчө басылат: адегенде Биринчи_клавишаны(Ctrl) басып, аны кое бербестен туруп Экинчи_клавишаны(K) жана Үчүнчү_клавишаны (B) удаалаш басуу керек.

3. Төмөнкүчө жазылуучу үч клавишанын комбинациясы

Биринчи_клавиша + Экинчи_клавиша + Үчүнчү_клавиша (мисалы, Shift+Ctrl+End)

төмөнкүчө басылат: адегенде бир мезгилде Биринчи_клавишаны (Shift) жана Экинчи_клавишаны (Ctrl) басып, аларды кое бербестен туруп Үчүнчү_клавишаны басуу керек.

Башкы менюнун каалаган камтылуучу менюсун тез ачуу

Ачылган менюнун командаларын тез чакыруу командалардын аттарында түс менен бөлүнүп көрсөтүлгөн тамгаларды клавиатурада басуу менен аткарылат. Ачылып турган File менюсунда «A» тамгасын бассак, анда Save as… командасы чакырылат.

Андан сырткары кээ бир командаларды тез чакырууну «ысык» клавишаларды басуу менен аткаруу мүмкүн. ТИЧтин «ысык» клавишаларынын тизмеси жогоруда келтирилген.

Жумушчу (учурдагы) каталогду өзгөртүү

1. **File** менюсун ачкыла
2. **Change dir …** командасын тандагыла. Диалогдук терезе ачылып анын **Direktory tree** талаасында учурдагы дисктин каталогдор дарагы көрсөтүлгөн.
3. **Direktory name** талаасында учурдагы жаңы каталогдун атын кийирүү ичинде же **tab** клавишасын баскандан кийин аны каталогдор дарагында көрсөткүлө.
4. Тандалган каталогду **Enter** клавишасын басуу менен фиксирлегиле.
5. Tab клавишасын басуу менен курсорду OK кнопкасына койгула жана **Enter** клавишасын баскыла.

Жаңы программаны кийирүү үчүн терезе ачуу

1. **File** менюсун чакыргыла
2. **New** командасын чакыргыла

Жаңы программаны дискке жазуу

1. **File** менюсун ачкыла.
2. **Save as…** командасын чакыргыла(тандагыла). **Save File As** диалогдук терезеси ачылат, анын **Save File As** талаасында жаңы программанын аты киргизилет. Бул терезеде жаңы программанын мурда бар болгон файлдын биринин аты менен (сактоо) жазуу мүмкүнчүлүгү да бар. Ал үчүн **Tab** клавишасын басуу (курсор **Files** талаасына коюлат) жана чыгып турган тизмеден файлдын атын тандоо керек.
3. Enter клавишасын баскыла. Бул учурда файлды мурда бар болгон ат менен жазганда тиешелүү эскертүү берилет жана файлды кайра жазуу же файлды тандалган ат менен жазуудан баш тартуу мүмкүнчүлүгү камсыз кылынат.

Редактирлөө үчүн файл ачуу

1. **File** менюсун ачкыла
2. **Open…** командасын чакыргыла. (F3 клавишасынын аракети ушул команданын аткарылышына эквиваленттүү).
3. Экранда **Open a File** терезеси пайда болот. **Name** талаасы **Files** талаасына чакырылуучу файлдарды тандоо үчүн шаблонду кармап турат. Көрсөтүлбөгөн учурда *.PAS шаблону сунуш кылынат. **Files** талаасында дисктин учурдагы каталогундагы шаблонго тиешелеш файлдын тизмеги чакырылат.

Активдүү терезедеги программаны эски ат менен сактоо

1-ыкма. F2 клавишасын баскыла
2-ыкма. 1. File менюсун ачкыла
2. Save командасын чакыргыла.

Активдүү терезедеги программаны аткаруу

Ctrl+F9 «ысык» клавишаларын баскыла

Жыйынтыктарды чыгаруу үчүн экранды тазалоо

Программанын ичинде экранды тазалоону ишке ашыруу үчүн төмөндөгүлөрдү аткаруу зарыл

- uses сүйлөмүндө Crt стандарттык модулун кошуу керек (uses Crt);
- программада оператордук блоктун башында ClrScr процедурасын чакыруу керек.

Мисалы:

```
program   Misal;
  uses Crt;
     var x, y, z : Integer ;
        begin    ClrScr;
              read (x,y);
              z:=x+y;
              writrln('z=' ,z)
         end.
```

2. СИМВОЛДОР ЖЫЙЫНДЫСЫ, ЛЕКСЕМАЛАР, АЖЫРАТКЫЧТАР

Аз бил бирок пайдалан,
Аз айт бирок аткар.
(Гете)

2.1. Символдор жыйындысы

Turbo Pascal тилинин символдорунун жыйындысы ASCII кодунун символдор жыйындысынын камтылуучу көптүгү болуп эсептелет.

- Латын алфавитинин кичине жана чоң тамгалары, ошондой эле тамгалар менен бирдей пайдаланылуучу астын сызуу символу;

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z **(ASC II коддору 65 тен 90 го чейин)**

a b c d e f g h i j k l m n o p q r s t u v w x y z **(ASC II коддору 97 ден 122 ге чейин)**

_(ASCII коду 95)

- Ондук араб цифралары;

0 1 2 3 4 5 6 7 8 9 **(ASCII коддору 48 ден 57 ге чейин)**

- Атайын символдор (кашаалар ичинде ASC II коду көрсөтүлгөн);

# (35)	– (45)	@ (64)
$ (36)	. (46)	[(91)
′ (39)	/ (47)	] (93)
((40)	: (58)	^ (94)
) (41)	; (59)	{ (123)
* (42)	< (60)	} (125)
+ (43)	= (61)	
, (44)	> (62)	

- Пробел символу (ASC II коду-32);
- Башкаруу символдору (ASC II коддору 0 дон 31 ге чейин).

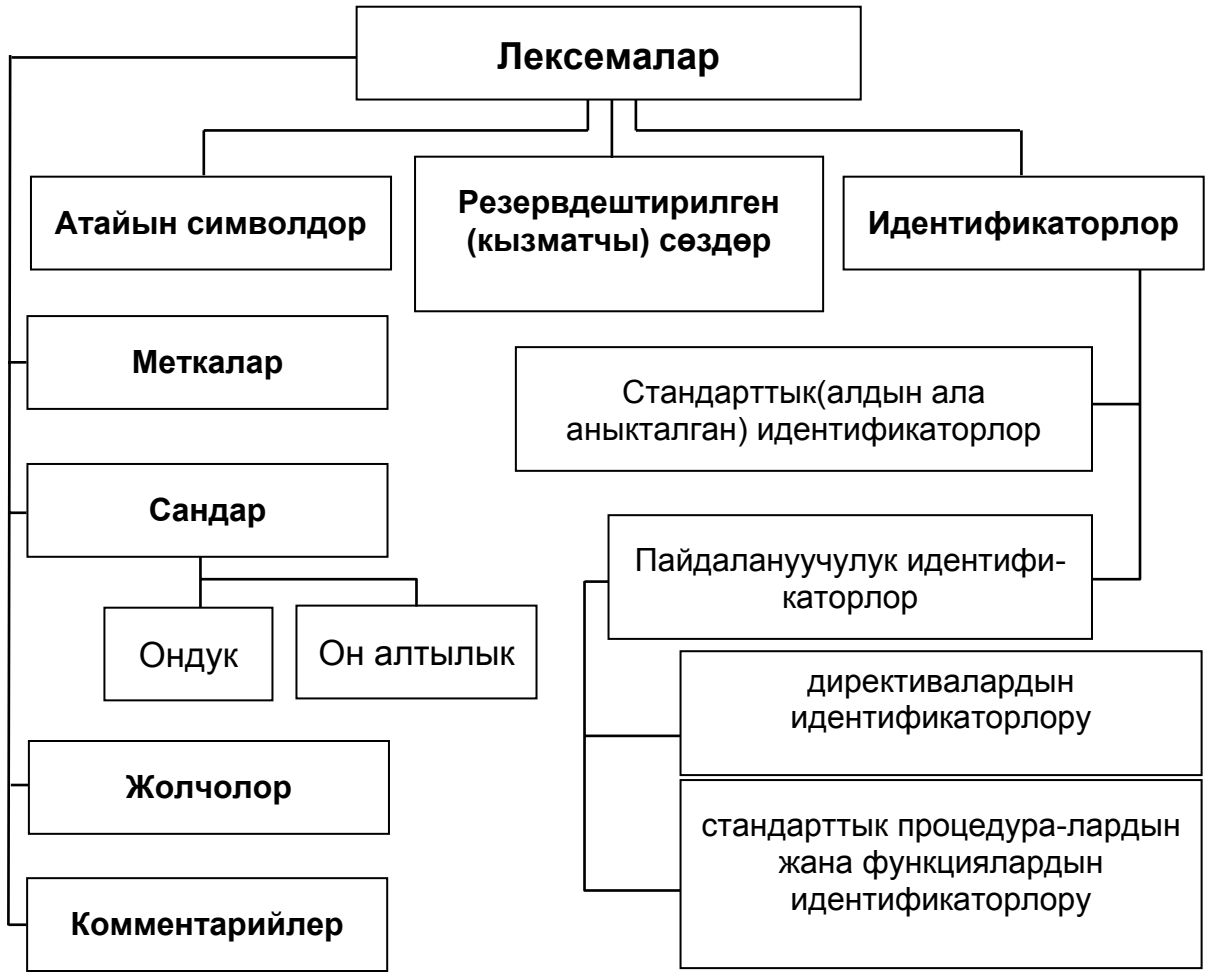
ASC II кодунун негизги жыйындысынын калган символдору жана ASCII кодунун кеңейтирилген жыйындысынын бардык символдору (кирилицанын тамгалары кошо) лексемаларды түзүү үчүн

да, ажыраткычтар катары да колдонулбайт. Бирок алар символдук жана жолчолук константаларда, ошондой эле пайдалануучу менен кампьютердин ортосундагы орус же кыргыз тилиндеги диалогду ишке ашырууга мүмкүнчүлүк берүүчү коментарийлерде пайдаланылат.

2.2. Лексемалар

Лексемалар - деп программанын текстиндеги мааниге ээ болгон (значимые) минималдык бирдиктерди аташат. Тактап айтканда, трансляция теориясында “символ ” жана “лексема” терминдеринин ортосунда, эреже катары, айырма жасалбайт. “Символ” терминининн ASCII кодунун символдоруна карата болгон байланышынын негизинде бул термин ушундай мааниде колдонулат. Ал эми тилдин грамматикасынын лексикалык бирдиктерин баяндоодо (идентификатор, кызматчы сөз ж.б.) “лексема” термини пайдаланылат.

Turbo Pascal илинде лексемалардын бир нече категорияларын бөлүп көрсөтүүгө болот, алар 2.1 - сүрөттө көрсөтүлгөн.



2.1 - сүрөт. Лексемалардын категориялары

2.2.1. Атайын символдор

Лексемалар болуп эсептелген атайын символдорго төмөнкүлөр кирет.

+ – / * < = > [ ] ( ) { } . , : ; ^ @ # $

Андан сырткары атайын символдордун түгөйлөрү(курама символдор) дагы лексемалар болуп эсептелет

:= .. <= >= (* *) (. .)

(* лексемасы {. лексемасына эквиваленттүү.
*) лексемасы }. лексемасына эквиваленттүү.
(. лексемасы [. лексемасына эквиваленттүү.
.) лексемасы]. лексемасына эквиваленттүү.

2.2.2. Резервделген (кызматчы) сөздөр

Резервделген сөздөр Turbo Pascal тилинде так аныкталган арналышка ээ болуп, аны өзгөртүүгө болбойт. Ошондуктан резервделген сөздөр менен окшош болгон пайдалануучулук идентификаторлорду баяндоого болбойт.
Turbo Pascal чоң жана кичине тамгалар менен (жогорку жана төмөнкү регистрлерде) жазылган сөздөрдү айырмалабайт.

Ошентип

program Program PROGRAM

сөздөрү бир жана бир эле резервделген сөздү билдирет. Бул касиет калган бардык резервделген сөздөргө жана программада пайдаланылуучу бардык идентификаторлорго таандык.

Turbo Pascal тилиндеги кызматчы сөздөрдүн жыйындысын келтиребиз.

and	**exports**	**mod**	**shr**
asm	**file**	**nil**	**string**
array	**for**	**not**	**then**
begin	**function**	**object**	**to**
case	**goto**	**of**	**type**
const	**if**	**or**	**unit**
constructor	**implementation**	**packed**	**until**
destructor	**in**	**procedure**	**uses**
div	**inherited**	**program**	**var**
do	**inline**	**record**	**while**
downto	**interface**	**repeat**	**with**
else	**label**	**set**	**xor**
end	**library**	**shl**	

Turbo Pacal 7.0 тилинде мурдагы версиялардан айырмаланып ***inherited*** *резервделген сөзү кошулган. Ал каралып жаткан объекттин түздөн-түз тегин (предка) көрсөтүү үчүн псевдоат (псевдоимя) катары пайдаланылат.*

Дагы белгилеп кетчү нерсе, Turbo Pascal 7.0 версиясына жаңы лексикалык киргизилген нерселер (Assigned функциясы, Break жана Continue процедуралары, public стандартык директивасы) резервделген сөздөр болушпайт. Ушул фактты ырастоо максатында көрсөтүлгөн лексемалар пайдалануучулук идентификаторлор катары пайдаланылуучу төмөнкү программаны карайбыз.

```
program TestResWord;
uses Crt;
var
Assigned, Break,
Continue, public:Byte;
Begin  ClrScr;
     Assigned:=1; Break:=2;
     Continue:=3; public:=4;
Writeln(assigned, ’  ’ , break, ’  ’ , continue, ’  ’ , public);
Writeln(‘Assign, break, continue, public лексемалары’);
Writeln(‘резервделген сөздөр болушпайт’);
end.
```

2.2.3. Идентификаторлор

Идентификатор – бул тамга менен же астын сызуу (_) символу менен башталуучу, пробелдерди кармабаган тамгалардын, цифралардын жана астын сызуу символдорунун удаалаштыгы болуп эсептелет. Резервделген сөздөр сыяктуу идентификаторлор дагы клавиатуранын жогорку жана төмөнкү регистрлерин айырмалабайт.

Идентификаторлор константалардын, типтердин, өзгөрүлмөлөрдүн, процедуралардын, функциялардын, модулдардын, програмалардын жана жазууларда талаалардын аттары катары кызмат кылат. Идентификатор каалагандай узундукка ээ боло алат, бирок маани берүүчү болуп биринчи 63 символ гана эсептелет.

Туура идентификаторлор	**Туура эмес идентификаторлор**
MyConst My_Type _Variable Non_Stop1 _1_3_Digital	MyVariable# - себеби уруксат берилбеген # символун кармап турат My-Type - себеби уруксат берилбеген - символун кармап турат 1_Variable - себеби цифра менен башталган

Өзгөчө учур болуп, качан ар түрдүү модулдарда бирдей атка ээ болгон элементтердин баяндалышы эсептелет. Мындай ситуацияда **квалификацияланган** (квалифицированные) **идентификаторлор** деп аталган идентификаторлор пайдаланылат. Мында өзгөрүлмөнүн атынын алдына ушул өзгөрүлмөнү кармап турган модулдун аты коюлат. Ошондо эки идентификатор чекит менен ажыратылат. Квалификацияланган идентификаторлорду дагы башкача **такталган**(уточненные) идентификаторлор деп аташат.

Мисал

Unit1.My_Var Unit2.My_Var Unit3.My_Var

Бул көрсөтүлгөн учурдан башка такталган идентификаторлор жазуулар жана объекттер менен иштөөдө пайдаланылат.

Кандай идентификаторлор такталган боло алат, а кайсылар боло албайт ал төмөнкү диаграммада көрсөтүлгөн.

метканын идентификатору	такталган идентификатор
константанын идентификатору	
типтин идентификатору	
өзгөрүлмөнүн идентификатору	
процедуранын идентификатору	
функциянын идентификатору	

программанын идентификатору	такталбаган идентификатор
модулдун идентификатору	
талаанын идентификатору	

Turbo Pascal тилинде идентификаторлордун эки түрү бар:

- стандарттык (алдын ала анакталган)
- пайдалануучулук.

Стандарттык алдын ала аныкталган идентификаторлор болуп тилдин ичине тиркелген бардык процедуралардын жана функциялардын (Read, Write, Sin ж.б.), типтердин (Integer, Real, Char ж.б.), директивалардын (forward, absolute, private, public ж.б.) аттары эсептелет. Стандрттык идентификаторлорду башка максатта пайдалануу үчүн кайра аныктоо мүмкүн, бирок бул учурда алардын стандарттык кызматы берилген программа үчүн жоголот. Ошондуктан мындайча кайра аныктоо программалодо жаман стиль деп эсептелинет.

Кээ бир стандарттык директиваларды, алардын колдонулуш спецификаларын эске алуу менен, дагы башкача процедуралык директивалар деп аташат.Turbo Pascal 7.0 тилиндеги стандарттык директивалардын жыйындысын келтиребиз.

absolute
assembler(процедуралык директива)
export(процедуралык директива)
external(процедуралык директива)
far(процедуралык директива)
forward(процедуралык директива)
index
interrupt(процедуралык директива)
near(процедуралык директива)
ptivate
public
resident
virtual(процедуралык директива)

*Мурдагы версиялардан айырмаланып жаңы **public** директивасы киргизилген, ал объекттин жалпы (интерфейстик) бөлүгүнүн башталышын билдирет.*

Turbo Pascal тилинин алтынчы версиясында киргизилген private директивасы менен бирге бул директива модулдун структурасына окшогон объектардин структурасын түзүүгө мүүмкүнчүлүк берет.

Жогоруда каралып өткөн эрежелерден сырткары, программаларды жазууда редактирлөөнү жана алардын көрсөтмөлүүлүгүн жогорулатууга мүмкүнчүлүк берүүчү төмөнкү сунуштарды эске алуу бир топ пайдалуу.

1) идентификаторлорду түзүп жатканда аларды ”маанилештирип” жасоого аракет кылуу керек , аларга келгенде эконом кылуунун кажети жок.

Албетте өзгөрүлмөгө берилүүчү My_Variable аты, MV же M_var атына караганда жакшыраак.

2) тилдин бардык структуралары англис тилиндеги идентификаторлор менен иш жүргүзөт, бирок программист-пайдалануучу өзүнүн жеке элементтери үчүн орус же кыргыз тилиндеги (бирок сөзсүз латын тамгалары менен жазылган) идентификаторлорун түзүп алса болот.

Мисалы

```
Program Salam;
 var
     tamyr1, tamyr2: real;
```

2.2.4. Эн-белгилер (Меткалар)

Turbo Pascal тилинде э**н-белгилердин** эки түрү бар: **сандык** жана **символдук**.

Сандык эн-белги бул **0** дөн **9999** га чейинки диапазондогу сандардын удаалаштыгы болушу мүмкүн. Сандын башталышындагы нөлдөр маани берүүчү болуп эсептелбейт. Сандык меткалар Pascal тилинин бардык версияларында реализацияланган. Turbo Pascal башка кээ бир кеңейтирилген реализациялар сыяктуу меткалар катары идентификаторлорду (аларды жазуу эрежелери жогоруда каралды) пайдаланууга мүмкүнчүлүк берет. **Эн-белги оператордон кош чекит (:) символу менен ажыратылат.**

Мисал. Программада A:=B*Sqrt(c) оператору c1 эн-белгисине ээ болсо, анда ал төмөнкүчө жазылат

```
c1: A:=B*Sqrt(c);
```

2.2.5. Сандар

Turbo Pascal тилинде бүтүн ондук, бүтүн он алтылык жана чыныгы ондук сндар пайдаланылат. Чыныгы ондук сандар өз кезегинде эки ар түрдүү формада көрсөтүлүшү мүмкүн:

- кадимки (фиксирленген ондук чекиттүү)
- көрсөткүчтүк (жылма ондук чекиттүү)

Бүтүн ондук сандар стандарттык көрүнүштө жазылат жана – 2147483648 ден 2147483647 ге чейинки диапозондо болушу керек.

Мисалы,

53 66 -48 6 -4567 234567

Он алтылык бүтүн сандарды белгилөө үчүн $ символу пайдаланылып, ал сандын алдына коюлат.

Мисалы

$0 $3E $FC45 $A10

Он алтылык сандардын уруксат берилген диапазону **$00000000** дөн **$FFFFFFFF** ге чейин. Он алтылык сандын белгиси (плюс же минус) жазылыш формасынын өзү менен аныкталат жана сандын экилик көрсөтүлүшүндөгү чоң разряддын маанисинен көз каранды болот.

Чыныгы сандар же кадимки ондук бөлчөк көрүнүшүндө, же 10 негизи боюнча көрсөткүчтүк формада жазылышы мүмкүн. Көрсөткүчтүк формада жазылган учурда 10 негизинин ордуна E тамгасы (чоң же кичине) коюлат жана андан соң түздөн-түз даражанын көрсөткүчү көрсөтүлөт.

Мисалы,

а) кадимки ондук бөлчөк көрүнүшүндө

5.12 6.3 -4136.281 36.59

б) көрсөткүчтүк формада

5.8E12 - 43.83e7 -0.2745e-10

Ондук чекиттүү же көрсөткүчтүк формадагы сандар чыныгы типтеги константалар болушат, ал эми калган ондук жана он алтылык сандар бүтүн типтеги константалар болушат.

2.2.6. Жолчолор

Символдор жолчосу ASCII кодунун кеңейтирилген жыйындысынан алынган бир кабат тырнакчалар ичине алынган символдордун удаалаштыгы болуп эсептелет. Ошону менен катар дагы бир зарыл болгон нерсе, мындай удаалаштык бүтүндөй түрдө программанын бир жолчосунда жайгашкан болушу керек.

Жолчонун курамына бир кабат тырнакча белгисинин өзүн символ катары кошуу үчүн бул тырнакча символу эки жолу катар жазылышы керек. Эгер тырнакчалардын арасында эч кандй символ жок болсо, анда мындай жолчо бош жолчо деп аталып, анын узундугу нөлгө барабарланат.

Мисалдар.

′Turbo Pascal 7.0′

′H.Bирт-Pascal тилинин автору′

′′- бул бош жолчо

′Эки катар келген тырнакча ″ жолчодо бир символ деп эсептелет′

Жолчодо башкаруучу символдор алдында түздөн-түз # символу коюлган ондук сан көрүнүшүндө көрсөтүлөт. Ал көрсөтүлгөн ондук сан талап кылынган башкаруучу символдун ASC2 коду болушу керек.

Мисалы,

#7-″коңгуро ″ символу

#10-″жолчону которуу” символу

#13- ″каретканы кайтаруу ″символу

Иш жүзүндө, мындай жол менен башкаруучу символдор гана эмес ASCII кодунун (0дон 255ке чейинки коддор) каалаган башка символу көрсөтүлүшү мүмкүн. Андан сырткары, жолчодо печаттык жана башкаруучу символдорду айкалыштырууга болот. Эгерде жолчодо бир нече башкаруучу символдор колдонулса анда алардын арасында ажыраткычтар болбошу керек.

Мисалы,

′#13#10′печаттык символдрунун жана
′#13#10′ башкаруу символдорунун айкалышышы

2.2.7. Комментарийлер

Комментарий – бул программада сол жагынан { символу же курама **(*** символу менен, ал эми оң жагынан } символу же курама ***)** символу менен чектелген тексттин фрагменти болуп саналат. Комментарийлер программаларда информациялык гана функцияны аткарышат жана кээ бир жекече камтылуучу программаларды , типтерди , констонталарды, өзгөрүлмөлөрдү ж.б. баяндап түшүндүрүү үчүн кызмат кылышат. Алар компилятор тарабынан четтетилет(игнорируется) да программанын ишине эч кандай таасирин тийгизбейт.

Төмөнкү конструкциялар комментарийлер болуп эсептелет.

{Эки жагынан фигуралык кашаалар менен чектелген символдордун каалагандай удаалаштыгы, канча жолчону ээлеп тургандыгына
каркарабастан комментарий боло алат}

(* ич жагынан жылдызчалар менен коштолгон тегерек кашаалардын
ичине алынган символдордун удаалаштыгы да комментарий болот *)

Өзгөчө учур болуп, { же (* ачуучу кашаасынан кийин дароо $-доллар белгиси келген коментарий эсептелет. Мындай комментарий компилятордун директивасы болуп саналат.

Мисалы,

{$N+} (*$R-*) {$I MyFile.pas}

2.3. Ажыраткычтар

Лексемаларды бири-биринен ажыратуу үчүн ажыраткычтар катары Turbo Pascal тилинин бардык версияларында төмөнкү символдор колдонулат;

- пробел(ASCIIкоду -32);
- табуляция(ASCIIкоду-09);
- кийинки жолчонун башталышына которуу курама символу (“ каретканы кайтаруу” (ASCII коду 13) жана “жолчону которуу” (ASCII коду10) символдорунун түгөйү).

Turbo Pascal тилинде бул «классикалык» ажыраткычтардан сырткары ASCII коддор жыйындысынын коду 0 болгон символдон баштап коду 31 болгон символго чейинки диапазондон алынган каалагандай башкаруучу символдор ажыраткычтар катары пайдаланышы мүмкүн.

Мисалы төмөкү программа толук иштөөгө жөндөмдүү. Анда program жана Test лексемаларынын ортосунда ажыраткыч катары коду 24 болгон башкаруучу символ, var жана A лексемаларынын ортосунда коду 25 болгон, ал эми begin жана A лексемалаларынын ортосунда коду 21 болгон башкаруучу символ турат.

```
Program ↑Test;
Var ↓A:I nteger;
Bedin A:=1;
Writeln(A);
End.
```

Мындай символдорду програмалардын тексттеринде пайдаланууга уруксат берилишине карабастан, программалодо аларды пайдалануу жаман стиль деп эсептелинет.

Ар кандай эки лексеманын ортосунда каалагандай сандагы символдорду-ажыраткычтарды коюга болот. Андан сырткары, ”атайын символдор” группасындагы лексемалар (2.1 - сүрөт) өздөрү ажыраткычтар болуп эсептелет, ошондуктан аларды башка лексемалардан ажыратуу үчүн символ-ажыраткычтарды пайдалануу шарт эмес. Жогорку мисалда мындай символдор болуп : , (,) символдору жана := курама символу эсептелет.

Тапшырмалар

1.Туура эмес идентификаторлорду көрсөткүлө:

а) Turbo	е) Sen_Men
б) _Year	ж) #53
в) 99_Yes	з) Asan@Usen
г) Turbo-2002	и) 2-AmericanBoy
д) 7TYC	к) _1Monday

2.Туура эмес сандарды көрсөткүлө:

а) –46578e+12	е) 0e2
б) 1e13	ж) 1,34
в) e8	з) 0.909
г) $2	и) .8
д) 9.	к) 2002$

3. ПРОГРАММАНЫН СТРУКТУРАСЫ

Ойсуз окуу пайдасыз, ал эми окуусуз ой коркунучтуу.
(Конфуций)

Turbo Pascal тилинде өз алдынча программа түзүүнү үйрөнүүнү каалаган пайдалануучулар үчүн бул тилде түзүлүүчү программанын структурасын жана анда сактоо керек болгон эрежелерди билүү чоң мааниге ээ болот. Turbo Pascal тилинде түзүлгөн ар кандай программаны шарттуу түрдө төрт бөлүккө бөлүүгө болот:

- Программанын бөркү;
- Пайдаланылуучу модулдарды көрсөтүү бөлүгү;
- Баяндоолор бөлүгү;
- Операторлор бөлүгү.

Ошентип, формалдык эмес түрдө программанын структурасын төмөнкүчө көрсөтүүгө болот.

```
{I. Программанын бөркү}
program Программанын_аты;
{II. Пайдаланылуучу модулдарды көрсөтүү бөлүгү}
 uses Пайдаланылуучу_модулдардын_тизмеси;
{III. Баяндоолор бөлүгү}
 label Меткаларды_баяндоо;
 Const Константаларды_ баяндоо;
 Type Типтерди_баяндоо;
 var Өзгөрүлмөлөрдү_баяндоо;
 procedure Процедураларды_баяндоо
 function Функцияларды_баяндоо
 export Экспорттолуучу_Аттарды_баяндоо;
{IV. Операторлор бөлүгү (Операторлук блок)}
begin
Операторлор
end.
```

3.1. Программанын бөркү

Turbo Pascal 7.0 тилиндеги программанын бөркү анын мурдагы версияларындагыдай эле сөзсүз жазылышы шарт эмес болуп ал көбүнчө декоративдик максатта пайдаланылат. Бирок, эгер баары бир ал программада катышып калса, анда аны синтаксистик жактан туура жазуу зарыл.

Мисалдар.

1.Program User1;
2. Program Print (Output);
3. Program Complex (Input, Ounput, MyFile);
4. Program Salam_saga;
5. Program Getput (Input, Output);

Бөрктүн сөзсүз шарт эмес экендигинен улам Turbo Pascal тилинде, эреже катары, 1-жана 4-көрүнүштөгү гана варианттар пайдаланылат.

3.2. Пайдаланылуучу модулдарды көрсөтүү бөлүгү (uses сүйлөмү)

Пайдаланылуучу модулдарды көрсөтүү бөлүгү **uses** резерв делген сөзү менен башталат. **uses** сүйлөмүнүн программада сөзсүз катышышы шарт эмес. Эгерде программада Turbo Pascal тилинин System модулунан башка стандарттык модулдарында же пайдалануучу тарабынан түзүлгөн модулдарда аныкталган константалар, типтер, өзгөрүлмөлөр, процедуралар же функциялар пайдаланылса, ал ушундай учурларда гана баяндалат.

Мисалдар:

```
uses  Crt,  Graph;
uses  Grt,  Graph,  Mylib,  Stack;
```

Ар бир жекече программада uses сүйлөмү бир жолу (бир ирет) гана баяндалган жана түздөн түз програманын бөркүнөн кийин жайланышкан болушу керек.

System модулу дайыма көрсөтүлбөстөн эле (по умолчанию) пайдаланыла берет, ошондуктан аны uses сүйлөмүндө көрсөтүүнүн кереги жок. Бул модуль файылды кийрүү-чыгаруу, жолчолорду иштеп чыгуу, жылма чекиттүү сандар менен болгон амалдар, эсти динамикалык бөлүштүрүү сыяктуу каражаттар үчүн кызмат кылат. Turbo Pascal тилинин калган Dos, Crt, Graph жана башка ушул сыяктуу модулдары автоматтык түрдө кошулбайт (не подключаются), аларды пайдалануу керек болгон учурда сөзсүз uses сүйлөмүндө көрсөтүү керек.

Трансляция убагында талап кылынган модулдун кодун издөөдө, компилятор uses сүйлөмүндө көрсөтүлгөн модулдун атын биринчи сегиз символго чейин кесет жана ага файлдын кеңейтирилишин кошот. Эгер компиляциянын негизги платформасы (1.2-сүрөттү кара) Dos болсо, анда .TPU кеңейтирилиши кошулат, эгерде Windows болсо, анда .TPW кеңейтирилиши кошулат. Эгер DOS тун корголгон режими болсо, анда файылдын кеңейтирилиши .TPP болот. Файылдын аты кесилгени менен uses сүйлөмүндө модулдун идентификатору толук көрсөтүлүшү керек.

3.3. Баяндоолор бөлүгү

Баяндоолор бөлүгү дагы мындан мурдагы бөлүктөр сыяктуу сөзсүз көрсөтүлүшү шарт эмес болгон бөлүк. Бирок бул бөлүктү пайдаланбастан туруп эң жөнөкөй программаларды гана жазуу мүмкүн.

Константаларды (const), типтерди (type), өзгөрүлмөлөрдү (var), процедураларды (procedupe), функцияларды (function) баяндоо жана экспорт (exports) бөлүкчөлөрү баяндоолор бөлүгүнүн алкагында каалагандай тартипте көп жолу кайталанышы мүмкүн. Бир гана нерсе төмөнкү эреженин аткарылышына көз салуу зарыл.

Эреже:

Эгерде кандайдыр бир B элементинин (константанын, типтин, өзгөрүлмөнүн, процедуранын, функциянын, экспорт тизмесинин) баяндалышында A элементи (константа, тип ж.у.с.) пайдаланылса, анда A элементи B элементинен мурда баяндалышы керек.

Экспортту баяндоо (exports) бөлүмчөсү компилятордун процессордун корголгон режимин пайдалануучу версиялары, б.а. BP.EXE, BPC.EXE, BPW.EXE версиялары тарабынан гана реализацияланат.

Баяндоолордун бирдей бөлүмчөлөрүн көп жолу пайдалануу жогорку эреженин талабын аткаруу жана ошондой эле программанын структураланышын жана окумдуулугун (читабельность) жогорулатуу сыяктуу зарыл болгон учурларда гана колдонулат.

Мисалы,

```
type        1 -маселечени (подзадача) чечүү
var                  үчүн баяндоолор
procedure
{-------------------------------------------------------------}
```

```
label                2-маселечени чечүү үчүн
const                 баяндоолор
{-----------------------------------------------------------------}

{-----------------------------------------------------------------}
const              N-маселечени чечүү
type               үчүн баяндоолор
function
```

3.3.1. Эн-белгилерди (Меткаларды) баяндоо

Эн-белгилер программанын каалаган операторунун алдында келиши мүмкүн болуп алар операторлордон кош чекит (:) менен ажыратылат. Эн-белги goto - өтүү оператору менен биргеликте да пайдаланылып андай учурда ал кош чекитсиз жазылат.

Мисалы,

```
label 1, Quit;
       . . .
      goto 1;
    . . . .
1:  a:=1;
      goto  quit;
      . . .
Quit: end.
```

goto оператору менен программада кайрылуу (шилтеме) жасалбаган эн-белгилерди пайдалануу маанисиздик болот, бирок ката болуп эсептелбейт. Дагы белгилеп коюучу нерсе, эн-белгилерди жана goto операторлорун пайдалануу көбүнчө структуралык программалоонун принциптерине каршы келет ошондуктан программаларда бул конструкцияларды колдонуудан мүмкүн болушунча арылуу зарыл.

3.3.2. Типтерди баяндоо

Turbo Pascal тилинде каалагандай берилгендер (б.а. константалар, өзгөрүлмөлөр, функциялардын маанилери ж.б.) өздөрүнүн типтери менен мүнөздөлөт. Берилгендердин типтери төмөндөгүлөрдү аныктайт:

- компьютердин эсинде берилгендердин көрсөтүлүш форматын;
- тандалган типке таандык болгон өзгөрүлмө же константа кабыл алышы мүмкүн болгон маанилердин көптүгүн;
- бул типке колдонууга мүмкүн болгон амалдардын көптүгүн.

Мисалы үчүн integer тибиндеги өзгөрүлмөнү сактоо үчүн компилятор оперативдик эсте эки байтты бөлөт, ал болсо эсте -32768 (=2^15) ден 32767(=2^15-1) ге чейинки диапазандогу бүтүн сан маанилерди жайгаштырууну камсыз кылат. Бул диапазондон чыгып кеткен (ашып кеткен) бүтүн сандар, ошондой эле чыныгы сандар эстин мындай бөлүнүп коюлган көлөмүндө сактала албайт.

Turbo Pascal 7.0 типтердин “көп” (“күчтүү”) бутакталган структурасына ээ болуу өзгөчөлүгүнө ээ болгон программалоо тили болуп эсептелет. Программада колдонулуучу бардык берилгендер берилгендердин кандайдыр бир мурдатан белгилүү болгон стандарттык (Turbo Pascal 7.0 дө берилгендердин алдын ала аныкталган (предопределенный) тиби) же пайдалануучулук (программист тарабынан алдын ала аныкталган тип) тибине таандык болушу керек.

Программада стандарттык типтеги берилгендерди пайдалануда аларды алдын ала аныктоонун зарылчылыгы жок. Ал эми берилгендердин пайдалануучулук (пользовательский) тиби мындай аныктоону талап кылат.

Ошентип Turbo Pascal тилинин типтер көптүгүн эки тайпага (группага) бөлүүгө болот:

- **стандарттык** (алдын ала аныкталган) типтер;
- **пайдалануучулук** (пайдалануучу тарабынан аныкталган) типтер.

Стандарттык типтердин аттары алдын ала аныкталган идентификаторлор болуп (резервделген сөздөр менен эч качан чаташтырбоо керек), программанын каалаган жеринде аракет кыла алат. Чындыгында алар System стандарттык модулунда баяндалып, бул модуль uses бөлүгүндө көрсөтүлгөн - көрсөтүлбөгөндүгүнөн көз карандысыз, атайын көрсөтүлбөгөн учурда (по умолчанию) ар бир программадагы пайдаланылуучу модулдардын жана ар бир пайдалануучунун модулдарынын тизмесине автоматтык түрдө кошулат (подключается). Башка алдын ала аныкталган идентификаторлор сыяктуу эле стандарттык типтердин аттары дагы кайрадан аныкталган болушу мүмкүн. Бирок мындан кийин деле System модулунун атын көрсөтүү менен квалификацияланууучу идентификатордун жардамында анын баштапкы маанисине кайрылууга болот.

Мисалы,

System.Integer System.Real System.Char.

Turbo Pasral тилинде **стандарттык типтер** болуп төмөнкү типтер эсептелет:

- бүтүн типтердин тайпасы (Shortint, Integer, Longint, Byte, Word);
- чыныгы типтердин тайпасы (Single, Real, Double, Extended, Comp);
- бульдук типтердин тайпасы (Boolean, ByteBool, WordBool, LongBool);
- символдук тип (Char);
- жолчолук типтер (String, Pchar);
- көрсөткүчтүк тип (Pointer);
- тексттик тип (Text).

Пайдалануучулук типтер – кошумча абстракттык (жөнөкөй жана структураланган) типтер болуп, алардын мүнөздөмөсүн програмист - пайдалануучу өз алдынча аныкташы мүмкүн. Мындай типтерди пайдалануу програмистке коюлган маселени ачык жана так баяндоого мүмкүнчүлүк берет, ал эми компиляторго синтаксистик каталарды текшерүү жана бир кыйла эффективдүү машиналык кодду генерациялоо үчүн көбүрөөк информация берет.

Пайдалануучулук типке төмөнкү типтер кирет:

- саналуучу тип;
- интервалдык тип;
- көрсөткүчтүк типтер (Pointer стандарттык тибинен башка);
- структураланган типтер;
- процедуралык тип.

Берилгендердин пайдалануучулук тибин жорыялоо type кызматчы сөзү менен ачылган типтерди жарыялоо бөлүгүндө жүргүзүлөт. type резерв делген сөзүнөн кийин бири-биринен барабардык белгиси менен ажыратылып жазылган жаңы пайдалануучулук типтин аты жана бул типти аныктоочу конструкция көрсөтүлөт.

Мисалы,

```
Type
        Day = (′ Шейшемби′, ′ Жума ′, ′ Жекшемби′);
        Month = (′Май′, ′Июнь′, ′Июль′, ′Август′);
```

Бул мисалда Day жана Month сөздөрү жаңы типтердин аттары болуп эсептелет жана мында, мисалы, Day тиби үч элемент менен (′Шейшемби′, ′Жума′, ′Жекшемби′ сөздөрү) аныкталган, алар болсо String - жолчо стандарттык тибине таандык.

Пайдалануучулук типти жарыялоодо алардын аттары менен аларды аныктоочу конструкциялардын ортосуна барабардык белгиси (=) коюлган. Бул белгини эч качан ыйгаруу (:=) белгиси менен чаташтырбоо керек.

Бизге белгилүү болгондой, өзгөрүлмөнүн тиби анын дакларациясы кезинде аныкталат. Ошону менен бул өзгөрмөнүн эске сакталыш форматы жана анын мүмкүн болгон маанилеринин диапазону аныкталат. Паскальдын базалык концепцияларынан бири - бул амалдарда типтердин тиешелештигин катуу текшерүүнү ишке ашырышы. Бул болсо программаларда каталарды издөөнү алда канча жөнөкөйлөтөт.

Turbo Pasral тилинде жогоруда каралган типтерди **иреттик** жана **иреттик эмес** типтер деп дагы эки тайпага бөлүүгө болот.

Иреттик типтер төмөнү төрт касиет менен мүнөздөлөт.

- Каалагандай иреттелген типтин кабыл алууга мүмкүн болгон маанилеринин көптүгү ар бир элементи иреттик номерге ээ болгон иреттелген удаалаштык болуп эсептелет. Иреттик номер бүтүн сан менен көрсөтүлөт. Ар кандай иреттелген типтин биринчи мааниси нөл деген иреттик номерге, экинчиси бир деген номерге ж.у.с. ээ болот. Integer, ShortInt жана LongInt иреттик типтери үчүн бул орун албайт. Анткени бул типтердин маанилеринин иреттик номери болуп ошол маанинин өзү эсептелет.
- Иреттик типтин калагандай мааниси үчүн анын иреттик номерин аныктап берүүчү Ord стандарттык функциясын колдонууга болот.
- Иреттик типтин каалаган мааниси үчүн бул мааниден мурда келүүчү маанини аныктап берүүчү Pred стандарттык функциясын колдонууга болот. Бульдук типтерден башка ар кандай иреттик типтин биринчи элементи үчүн бул функция колдонулса, анда ал акыркы маанинин иреттик номерин аныктап берет.
- Иреттик типтин каалагандай мааниси үчүн бул элементтен кийин катар келүүчү элементтин маанисин аныктап берүүчү Succ стандарттык функциясын колдонууга болот. Бульдук типтерден башка каалагандай типтер үчүн, эгер анын акыркы элементине бул функция колдонулса, анда ал типтин биринчи маанисинин иреттик номерин берет. Стандарттык типтердин ичинен символдук, ошондой эле бульдук жана бүтүн типтер иреттик типтер болуп эсептелет.

Иреттик типтердин касиеттерин, ошондой эле Pred жана Succ функцияларынын ишин төмөнкү программа аркылуу демонстрациялоого болот.

```
program TestStandardOrdinalTypes;
  uses crt;
  var  {бүтүн типтер}
        i: Integer; si: shortint;  li: Londit;
              bt: Byte;  w: Word;
```

```
      {булдук типтер}
      b:  Boolean;
          wb: Wordlool; bb: ByteBool; lb: LongBool;
          {символдук тип}
          c: Char;
   begin
     ClrScr;
     i=10;  si:=0;  li:=-30;  bt:=255;  w:=0;
     Writeln (′Integer:     ′, Pred(i) :5, i:5, Succ(i):5);
     Writeln (′Shortint:  ′, Pred(si) :5, si :5, Succ(si) :5);
     Writeln (′Longint:   ′, Pred(li) :5, li :5, Succ(li) :5);
     Writeln (′Byte:  ′,  Pred(bt) :5, bt :5, Succ(bt) :5);
     Writeln (′Word: ′, Pred(w) :5, w :5, Succ(w):5);
     Writehn;
      {-------------------------------------------------------------------}
        b :=True;  wb :=False;  bb :=True;  lb :=False;
       Writeln (′Boolean : ′, Pred(b) :7, b :7, Succ(b):7);
       Writeln (′WordBool : ′, Pred(wb) :7, wb :7, Succ(wb) :7);
       Writeln (′ByteBool : ′, Pred(bb) :7, bb :7, Succ(bb) :7);
            Writeln (′LongBool : ′,Pred(lb):7,lb:7,Succ(lb):7);
            WriteLn;
  {-------------------------------------------------------------------}
     C:='b';
    Writeln(′Char :′, Pred(c) :5, c :5, Succ(c) :5);
     WriteLn;
end.
```

```
Программанын жыйынтыгы:

Integer:    9        10       11
Shortint:   -1       0        1
Hongint:    -31      -30      - 29
Byte:       254      255      0
Word:       65535    0        1

Boolean:    False    True     True
Wordboot:   True     False    True
Byteboot:   False    True     True
Longboot:   True     False    True
Char:       a        b        c
```

Эми ушул типтерге токтолуп өтөлү. Оболу стандарттык типтерди карайлы.

3.3.2.1. Стандарттык типтер

Бүтүн типтердин тайпасы

Turbo Pasral тилинде беш алдын ала аныкталган бүтүн-сандык типтер бар: ShortInt (кыска бүтүн), Integer (бүтүн), Longint (узун бүтүн), Byte (байт узундугундагы) жана Word (сөз узундугундагы). Бул типтерге таандык болгон элементтердин кабыл алууга мүмкүн болгон маанилеринин диапазону, аларды жайгаштыруу үчүн талап кылынган эстин областы жөнүндөгү маалыматтар төмөнкү таблицада көрсөтүлгөн.

Тип	Көрсөтүлүш диапазону	*Эстин өлчөмү*
ShortInt	**-128..127**	**1байт**
Integer	**-37768..32767**	**2байт**
LongInt	**-2147483648..2147983647**	**4байт**
Byte	**0..255**	**1байт**
Word	**0..65535**	**2 байт**

Бүтүн-сандык типтердеги (мисалы Integer, LongInt ж.б.) ар түрдүү өзгөрмөлөрдүн жогорку чек аралык мааниси константа катары берилген жана тиешелүү атка ээ:

$$MaxInt = 32767 = 2^{15}-1$$
$$MaxLongInt = 2147483647 = 2^{31}-1$$

Программанын текстинде бүтүн-сан маанилер пайдалануучу үчүн көнүмүш болгон формада жазылат.

Мисалы, 5300 жана – 25 маанилерин төмөнкүчү жазууга болот.

5300 (же +5300) жана –25

Бүтүн-сандык типке таандык болгон ар кандай өзгөрүлмө программанын текстинде ачык көрсөтүлгөн болуп өзүнүн курамында ондук чекитти кармап турбашы керек б.а. төмөнкүдөй жазуулар ката болуп эсептелет.

53 E+2 же –25.0

Бүтүн-сандык маанилерди берилгендердин он алтылык форматында жазууга мүмкүн. Эгер программанын текстинде он алтылык сандарды пайдалануну кааласак, анда клавиатурадан ($) символун кийрип андан кийин түздөн-түз сандын өзүн кийрүү зарыл.

Мисалы,

$F1, $AD1F0, -$70, +$DC

Чыныгы типтердин тайпасы

Чыныгы сандар бүтүн-сандык өзгөрүлмөлөрдө сактала албайт жана алардын маанилери катары көрсөтүлө алышпайт. Мындай сандар үчүн Turbo Pascal тилинде берилгендер тибинин өзгөчө тайпасы - Real тайпасы жашап анда беш стандарттык чыныгы тип аныкталган: чыныгы (Real), жеке (одинарный) тактыктагы чыныгы (Single), кош тактыктагы чыныгы (Double), жогорку тактыктагы (Extended), татаал чыныгы же чыныгы форматтагы бүтүн (Comp). Бул типтердин мүнөздөмөлөрү төмөнкү таблицада көрсөтүлгөн.

Тип	Көрсөтүлүш диапазону	Мантиссанын маани берүүчү цифралары	Эстин өлчөмү
Single	**$1.5*10^{-45}$ .. $3.4*10^{38}$**	**7-8 белги**	**4 байт**
Real	**$2.9*10^{-39}$ .. $1.7*10^{38}$**	**11-12 белги**	**6 байт**
Double	**$5.0*10^{-324}$..$1.7*10^{308}$**	**15-16 белги**	**8 байт**
Extended	**$3.4*10^{-4932}$ .. $1.1*10^{4932}$**	**19-20 белги**	**10 байт**
Comp	**$-2^{63}+1$ .. $2^{63}-1$ же жакындаштырылгын түрдө -9.2*10..9.2*10^18**	**13-20 белги**	**8 байт**

Типтердин бул тайпасында бүтүн жана чыныгы типтердин өзгөчө “гибриди” болгон Comp тибин бөлүп көрсөтүү зарыл. Бир жагынан бул типтеги өзгөрүлмөлөр жана константалар бүтүн маанилерге гана ээ боло алат (бул бүтүн типтердин касиети). Экинчи жактан, бүтүн типтерден айырмаланып Comp тиби иреттик болуп эсептелбейт (бул чыныгы типтердин касиети). Ушул иреттик эмес болуу касиети Comp тибин чыныгы типтердин тайпасына кошууга алып келет. Бул типтердин ар бири тарабынан мантиссанын көрсөтүлүш тактыгын төмөнкү программа аркылуу демонстрациялоого болот.

```
prodram  TestRealTypes;
   uses  Crt;
   var
   r : Real ;  s : Single;  d: Double;
  e : Extended;   c : Comp;
   bedin   ClrScr;
     s:=1.23456789012345678 90e-30;
     r:=1.23456789012345678 90e-30;
    d:=1.23456789012345678 90e-30;
    e:=1.23456789012345678 90e-30
    c:=1.23456789012345678 90e-30;
```

```
    {-------------------------------------------------------}
  Writehn ( ′ Single: ′, s: 32);
  Writehn (′ Real: ′, r: 32);
  Writehn (′ Double: ′, d: 32)
  Writehn (′ Extended: ′e: 32);
  Writehn (′ Comp: ′, c: 32);
    end.
```

Бул программада ар бир чыныгы типтеги өзгөүлмөгө жогорку тактыктагы бөлчөк константа ыйгарылат, андан кийин өзгөрүлмөлөрдүн маанилери печатка чыгарылат. Натыйжада төмөнкүдөй жыйынтыктар алынат:

```
Sindle   :1.23456792604715494E-0030
Real     :1.2345678901236917 2E-0030
```

```
Double     :1.2345678901234568 2E-0030
Extended   :1.2345678901234567 9E-0030
Comp       :0.0000000000000000E+0000
```

Бул жыйынтыктардан көрүнүп тургандай, адегенде Single тибиндеги өзгөрүлмө үчүн, андан кийин Real тибиндеги, ошондой эле кийинки типтер үчүн константанын көрсөтүлүшүнүн корректтүүлүгү бузулат. Ал эми Comp тибиндеги өзгөрүлмө болсо санды жакынкы бүтүн санга чейин тегеректеп коет.

Turbo Pasral чыныгы типтеги сандардын үстүнөн амалдарды аткаруу үчүн кодду генерациялоонун эки моделин колдонот:

- аппараттык (80x87 соопроцессору болгон учурда)
- программалык (80x87 соопроцессору жок болгон учурда)

Тиешелүү моделди тандоо компилятордук $N жана $E директиваларынын жардамында ишке ашырылат.

Атайын көрсөтүлбөгөн үчурда (по умолчанию) коюлуучу {$N-} абалында бир гана Real чыныгы тиби менен эсептөөлөрдү аткаруучу код генерацияланат.

{$N+} болгон учурда болсо 80x87 сандык соопроцессорунун жардамында чыныгы типтердин үстүнөн бардык эсептөөлөрдү аткаруучу код генерацияланат. Директиванын мындай коюлушу (установкасы) бардык беш чыныгы типти тең колдонууга мүмкүнчүлүк берет.

Түздөн-түз 80x87 соопроцессору менен иш алып баруудан сырткары Turbo Pascal тилинде эгер соопроцессор жок болсо, анын

аракетин автоматтык түрдө эмуляциялоочу, аткаруу убактысынын библиотекасын (библиотека времени выполнения) ишке кошуу (подключение) мүмкүнчүлүгү бар. Бул библиотеканы ишке кошулушун компилятордун $E директивасы башкарат. Белгилей кетүүчү нерсе, $E директивасы Windows режиминде чыгуу кодун (выходной код) генерациялоодо ошондой эле жалпы компоновкасыз жеке модулдарды компиляциялоодо иштебейт.

{$E+} абалында Turbo Pascal дын компилятору компьютерде соопроцессор орнотулганбы же жокпу ошону текшерет. Эгер соопроцессор бар болсо, анда компилятор тарабынан генерацияланган код аны пайдаланат. Эгер соопроцессор орнотулбаган болсо, анда компилятордун чыгуу коду анын ишин эмуляциялайт.

{$E-} абалында 80x87 соопроцессорунун эмуляциясын колдонбоочу (не подерживающий) код генерацияланат.

Андан сыркары $N жана $E директиваларын биргелештирип колдонуунун өзгөчөлүктөрү жөнүндө айтууга болот. Эгерде {$N+, $E+} директивалары экөө тең орнотулган болсо, анда компьютерде соопроцессордун орнотулган-орнотулбагандыгынан көз карандысыз бир кыйла тез аракет этүүчү чыгуу коду генерацияланат. {$N+, $E-} абалында компилятор 80x87 соопроцессору менен түздөн-түз эсептөөлөрдү аткаруучу библиотеканы гана кошот. {$N-, $E+}, {$N-, $E-} абалдарында $E директивасы четтетилет.

Логикалык (Бульдук) типтердин тайпасы

Pascal тилинин мурдагы бардык версиялары, элементтери эки гана маанини: True(чын) жана False(жалган) – кабыл алышы мүмкүн болгон бир гана Boolean бульдук тибин камтыган. Turbo Pasral 7.0 версиясында дагы үч бульдук тип, ByteBool, WordBool жана LongBool типтери кошулган.

Бульдук типтердин мүнөздөмөлөрү төмөнкү таблицада келтирилген.

Типтин идентификатору	False маанисине туура келет	True маанисине туура келет	Эстин өлчөмү
Boolean	0 саны	Каалагандай 0 эмес сан	1 байт
ByteBool	0 саны	Каалагандай 0 эмес сан	1 байт
WordBool	Эки байтта теё 0 саны	Каалагандай 0 эмес сан	2 байт
LongBool	Бардык байтта 0 саны	Каалагандай 0 эмес сан	4 байт

Жаңы бульдук типтер талдануучу программалардын, False маанисине 0 саны, ал эми True маанисине 0 дөн айырмалуу каалагандай сан тиешелеш келүүчү Widows каптамасы (оболочкасы)

менен биргелешүүчүлүгүн камсыз кылуу үчүн киргизилген. Салыштыруу жана логикалык амалдарынын жыйынтыктарынын тиби болуп мурдагыдай эле Boolean тиби кала берет.

Тиркелген Ord, Pred жана Succ функцияларынын “эски” Boolean жана жаңы бульдук типтердеги өзгөрүлмөлөр менен болгон ишин төмөнкү программа менен демонстрациялайбыз.

```
Program Test1_BoolTypes;
{Бул тест Ord, Pred, Succ функцияларынын ар бир бульдук тип менен}
{ болгон ишин бул функциялар пайда кылуучу маанилерди түздөн-түз}
{ Writeln процедурасы аркылуу чыгаруу менен демонстрациялайт}
uses Crt;
{$R+}
var
 b: Boolean; bb: ByteBool;
 wb: WordBool; lb: LongBool;

procedure PrintBooleans;
 begin
 Writeln (‘Boolean Pred(b) b Succ(b)’);
 Writeln (‘Маани: ’ , Pred(b) : 9 , b:9, Succ(b) :9);
 Writeln (‘Ord() : ’ , Ord(Pred(b)) : 9 , Ord(b) : 9,
 Ord(Succ(b)): 9);
 Writeln (‘ByteBool Pred(bb) bb Succ(bb)’);
 Writeln (‘Маани: ’ , Pred(bb) : 9 , bb:9 , Succ(bb) :9);
 Writeln (‘Ord() : ’ , Ord(Pred(bb)) : 9 , Ord(bb) : 9,
 Ord(Succ(bb)): 9);
 Writeln (‘WordBool Pred(wb) wb Succ(wb)’);
 Writeln (‘Маани: ’ , Pred(wb) : 9 , wb:9 , Succ(wb) :9);
 Writeln (‘Ord() : ’ , Ord(Pred(wb)) : 9 , Ord(wb) : 9,
 Ord(Succ(wb)): 9);
 Writeln (‘LongBool Pred(lb) lb Succ(lb)’);
 Writeln (‘Маани: ’ , Pred(lb) : 9 , lb:9 , Succ(lb) :9);
 Writeln (‘Ord() : ’ , Ord(Pred(lb)): 9 , Ord(lb): 9,
 Ord(Succ(lb)) : 9);
 end;

 begin
 ClrScr;
 b:= False ; wb:= False ; bb:= False ; lb:= False;
 Writeln (‘Баштапкы маани: False’ );
 PrintBooleans;
 Writeln (‘Баштапкы маани: True’ );
 b:= True ; wb:= True ; bb:= True ; lb:= True;
 PrintBooleans;
end.
```

Программанын жыйынтыгы

```
Баштапкы маани :  False
Boolean          Pred(b)        b          Succ(b)
Маани:           True           False      True
Ord()   :        -1             0           1
ByteBool         Pred(bb)       bb         Succ(bb)
Маани:           True           False      True
Ord()   :        -1             0           1
WordBool         Pred(wb)       wb         Succ(wb)
Маани:           True           False      True
Ord()   :        -1             0           1
LongBool         Pred(lb)       lb         Succ(lb)
Маани:           True           False      True
Ord()   :        -1             0           1
Баштапкы маани :  True
Boolean          Pred(b)        b          Succ(b)
Маани:           False          True       True
Ord()   :        0              1           2
ByteBool         Pred(bb)       bb         Succ(bb)
Маани:           False          True       True
Ord()   :        0              1           2
WordBool         Pred(wb)       wb         Succ(wb)
Маани:           False          True       True
Ord()   :        0              1           2
 LongBool        Pred(lb)       lb         Succ(lb)
Маани:           False          True       True
Ord()   :        0              1           2
```

```
program  Test2_BoolTypes;
{
}
{ }
{ }
```

Бул мисалдардан көрүнүп тургандай Ord, Pred жана Succ функциялары бардык бульдук типтер менен бирдей иштейт. Мында Ord функциясы False мааниси үчүн 0 иреттик номерин берет, ал эми True мааниси үчүн тиешелүү бульдук өзгөрүлмөгө жазылган сандын ондук маанисин берет. Жаңы бульдук типтердин классикалык Boolean тибинен болгон айырмасы Pred жана Succ функцияларынын маанилерин тиешелүү типтердеги өзгөрүлмөлөргө ыйгаргандан кийин байкалат. Бул учурда Ord функциясы Boolean тибиндеги өзгөрүлмө үчүн True мааниси болгон учурда мурдагыдай

эле ушул өзгөрүлмөгө жазылган сандын ондук маанисин берет. Ал эми ByteBool, WordBool жана LongBool типтериндеги өзгөрүлмөлөр үчүн True мааниси учурунда Ord функциясы дайыма 1 деген маанини берет. Дагы бир байкай кетүүчү нерсе, компилятордун директивасы $R кандай коюлбасын бардык ыйгаруларды аткаруу маанилердин уруксат берилген (допустимый) диапазонунан чыгып кетүү катасын пайда кылбайт.

Символдук тип

Символдук типке Char стандарттык идентификатору тиешелеш коюлуп, ал бир символду (тамганы, белгини же кодду) сактоо үчүн арналган берилгендердин тиби болуп саналат. Бул типтеги өзгөрүлмөлөр жана константалар ASCII кодунун символдор көптүгүнөн маанилерди кабыл ала алат. Керек болгон символдун кодунун маанисин Ord функциясынын жардамында алууга болот, ал эми берилген код боюнча символду аныктоо Chr функциясы менен аткарылат. Андан сырткары, Char тибиндеги маанилерге башка иреттик типтеги маанилер сыяктуу Pred жана Succ функцияларын колдонууга болот.

Char тибиндеги өзгөрүлмө Byte тибиндеги өзгөрүлмө сыяктуу эле эсте бир байтты ээлейт. Turbo Pascal 7.0 тилинде берилгендердин бул тиби бир нече өзгөчөлүктөргө ээ. Аларды биз төмөнкү мисалда иллюстрациялайбыз.

```
program Character_Demo; {Char тибинин өзгөчөлүктөрү}
    var  ch: Char;
begin
    ch: = ′A′;          {ch өзгөрүлмөсүнө ′A′символу ыйгарылат  }
   Writeln(ch);         {′A′ символун экранга чыгаруу            }
   Writeln(Ord(ch)); {Ord функциясы ′A′ литерасынын ASCII     }
                        { кодун берет                            }
    ch:=#65;            { ch өзгөрүлмөсүнө ASCII коду 65 болгон  }
                        { символ ыйгарылат б.а. ′A′ символу      }
   Writeln (ch);        {′A′ символун экранга чыгаруу            }
     ch:=^a;            {Clrl+A комбинациясына тиешелеш келген }
                        { ABCII  код, ал код 1                   }
   Writeln (ch);        {“Кара тумшукчаны” чыгаруу,              }
                        {анын ASCII коду      1                  }
   Writeln (Ord(ch));  {^a га туура келген ASCII кодду чыгаруу   }
   Write (^G^J^M, ′кийинки жолчо′, ^G^G^J^M, ′жана дагы
          бир жолчо!!′);
end.
```

Мында Char тибиндеги өзгөрмөлөр үчүн маанилер символдордун жолчосу сыяктуу апострофтордун ичинде көрсөтүлөт. Андан сырткары түздөн-түз ASCII кодунун сандык маанисин көрсөтүү менен маанилерди берүү мүмкүнчүлүгү бар. Бул учурда символдун ASCII кодун билдирген сандын алдына биздин мисалда көрсөтүлгөндөй # белгисин коюу керек. Ошондой эле Char тибиндеги өзгөрүлмөлөрдүн маанилери катары көбүнчө “башкаруучу коддор” деп аталган атайын символдорду көрсөтүгө болот. Аларды (^) белгисинин жардамында аны менен катар литераны жазуу менен көрсөтүшөт. Башкаруучу коддорго ASCII кодунун толук аныкталган маанилери туура келет, а бул болсо, мисалы (^a) нын ордуна #1 ди кийрүүгө мүмкүндүк берет, анткени Ctrl+A нын коду 1 ге барабар. Башкаруучу коддор форматтап чыгаруу үчүн Writeln тиркелген процедуралары тарабынан көмүскө (бекитилген) түрдө колдонулат. Андан сырткары, программист өзү айкын түрдө чыгаруучу информациялардын агымына, алардын экранда жайгашышын башкаруу менен башкаруучу коддорду (биздин мисалда көрүнгөндөй) кошо алат. Ошондой эле информацияларды чыгаруу процессин алдын алуучу “үн коштоосун” түзүүгө болот.

Жолчолук тип

Берилгендердин структурасы көз карашы менен алып караганда жолчолук типтер башка типтерден айырмаланып структуралык типтерге жакын. Бирок жолчолордун Turbo Pascal тилинде прогаммалоодо кеңири колдонулушунан улам алар үчүн String жана PChar стандарттык типтери киргизилген. Ошентип бул типтерди жөнөкөй жана структуралык типтердин ортосундагы аралык типтер деп эсептесек болот.

String тибиндеги жолчолор менен иштөө Turbo Pascal дын бардык версияларында, ошонун ичинде TP 7.0 дө да ишке ашырылып алар бири-биринен эч нерсеси менен айырмаланбайт.

Мындай типтеги өзгөрүлмө символдордун чынжыржасынан турат, б.а. Char тибиндеги элементтерден турат. Жолчолорду стандарттык Write жана Writeln процедураларынын жардамы менен экранга чыгарууга, Read жана Readln стандарттык процедураларынын жардамында кийрүүгө болот.

Көпчүлүк учурларда String тибиндеги өзгөрүлмөлөр бир нече символдордон турган сөздөрдү жана билдирүүлөрдү сактоо үчүн пайдаланылат.

Turbo Pascal тилинин чөйрөсүндө String тибиндеги берилгендер менен амалдардын бүтүндөй бир тобу байланышкан. Алар жөнүндө өзгөчө сөз кылууга болот.

String тибиндеги өзгөрүлмөлөр төмөндөгүчө жарыяланган болушу мүмкүн.

```
var
    Character_ srting_ 1: String;
    Character_ srting_ 2: String[20];
    Character_ srting_ 3: String[255];
```

String тибиндеги өзгөрүлмө, эреже катары,өзгөрүлмөнүн атын, String резерврделген сөзүн жана чарчы кашаанын ичинде ал өзгөрүлмөгө берилиши мүмкүн болгон жолчонун узундугун көрсөтүү аркылуу жарыяланат. Эгер жолчонун максималдык узундугу көрсөтүлбөсө анда ал автоматтык түрдө 255 ке барабар деп алынат. Бул болсо жолчонун мүмкүн болгон максималдык узундугу.

PChar *тибиндеги жолчолор Turbo Pascal 7.0 версиясындагы жаңы киргизилген элемент болуп эсептелет*

PChar тиби бул жолчолордун аяктоо белгиси катары 0 (нөл) коддуу символ кызмат кылган жана нөл менен аяктоочуу жолчолор же ASCIIZ - жолчолор деп аталган жолчолордун көрсөтүлүш форматы менен иш жүргүзөт. Мындай жолчолор Windows да пайдаланылат, ошондуктан Windows дун башкаруусу алдында иштей турган программаларды түзүүнү жеңилдетүү үчүн Turbo Pascal да ASCIIZ - жолчолор кошулган (кийрилген).

Дагы бир белгилөөгө зарыл болгон нерсе, Pchar тиби берилгендердин жолчолук типтери менен иш жүргүзүү үчүн кийрилип иш жүзүндө ал

```
type Pchar = ^Char;
```

баяндоосуна ээ болгон **көрсөткүчтүк** тип болуп саналат.

Бул типтеги өзгөрүлмөлөр (компилятордун {$X+} директивасы орнотулган учурда)

```
array [0..K] of Char;
```

тибиндеги массив катары иштелинип чыгылат. Бул жерде K - аяктоо белгиси болгон символду эсепке албагандагы жолчодогу символдордун саны.

Көрсөткүчтүк тип

Стандарттык көрсөткүчтүк тип Pointer идентификаторуна ээ болот. Көрсөткүчтүк типтеги өзгөрүлмөлөрдүн жана константалардын маанилери болуп сегменттин адресинен (1сөз = 2 байт) жана жылыштан (смещение) (1сөз= 2байт) турган оперативдик

эстин адрестери эсептелет. Толук адрес эки сөздөн туруп, сементтин адреси анын чоң сөзүндө, ал эми жылыш кичине сөзүндө сакталып турат.

Pointer тибиндеги элементтер пайдалануучу тарабынан аныкталуучу көрсөткүчтүк типтерден айырмаланып каалаган типтеги өзгөрүлмөнүн адресин кармап тура алышат.

Тексттик тип

Text стандарттык тексттик тиби тексттик файлдарды иштеп чыгуу үчүн арналган. Бул тип файлдарга арналган темада каралган.

Эми берилгендердин пайдалануучулук типтерин кароого өтөлү.

3.3.2.2. Пайдалануучулук типтер

Пайдалануучулук типтердин ичинен саналуучу жана интервалдык типтер иреттик типтер болуп саналат. Ошондуктан бул типтерге жогоруда биз карап өткөн иреттик типтердин касиеттери таандык. Калган пайдалануучулук типтер иреттик боло алышпайт.

Саналуучу типтер

Саналуучу типтер санап өтүү жолу менен берилген идентификаторлордун иреттелген жыйындысы катары аныкталат. Мында идентификаторлордун тизмеси (үтүр менен ажыратылып) тегерек каашалардын ичинде көрсөтүлөт. Мисалы, WEEKDAY (аптанын күнү) саналуучу тиби Monday (Дүйшөмбү), Tuersday (Шейшемби), …, Sunday (Жекшемби) идентификаторлорун кармап турат. Turbo Pascal 7.0 программалоо тилинде WEEKDAY жана SEASON саналуучу типтерин жарыялоо төмөндөгү көрүнүштө болот:

```
TYPE
SEASON = (Spring, Summer, Autumn, Winter);
WEEKDAY = (Monday, Tuesday, Wednesday, Thusday,
                                  Friday, Satuday, Sunday);
```

Бул мисалдан көрүнүп тургандай мындагы ар бир элемент уникалдык идентификатор болуп эсептелет.

Төмөндөгүдөй жазуу менен аныкталган ар кандай саналуучу тип үчүн,

```
Type T= (Иденгтификатор W(1),
                Идентификатор W(2), …, идентификатор W(n));
```

мында T - типтин идентификатору, төмөндөгүлөр туура болот:

1). W(I) идентификатору <> W(J) идентификаторуна, эгер I<>J болсо.
2). W(I) идентификатору < W(I) идентификатордон, эгер I<J болсо.
3). T тибинин маанилери болуп W(1), W(2), …, W(N) идентификаторлору гана келет.

Саналуучу типтер логикалык туюнтмаларда да колдонулушу мүмкүн:

Мисалы,

```
IF MONTH = MAY THEN Writeln('Май');
```

Ошондой эле аларга салыштыруу амалдарын колдонуу мүмкүн:

Мисалы,

```
IF DAY > Friday THEN Writeln('Дем алыш күн');
```

Эми берилгендердин саналуучу тиби колдонулган мисалды карайлы.

```
Program  Demo_uses_enimeration_type;
   type
   MyInteger =   integer;
   MyReal     =   real;
        Week  =   (Monday, Tuersday, Wednesday,
   Thursday, Friday, Saturday, Sunday);
   Sex        =    (Masculine familine);
    Var Day : Week;
      begin
           Day := Thursday;
            if  (Day = Sunday) or (Day = Saturday)
                     then Writeln('Уикенд')
                     else Writeln('Аптанын жумуш куну');
      end.
```

Эң оболу көңүл бурчу нерсе, бул программада алгач стандарттык integer жана real типтерине тиешелеш келүүчү берилгендердин жаңы MyInteger жана MyReal типтери жарыяланат. Андан кийин берилгендердин Week (аптанын күндөрү) жана Sex (жынысы) саналуучу типтери жарыяланат. Бул типтер алардын мүмкүн болгон бардык маанилерин санап көрсөтүү менен аныкталат.

Саналуучу тип жарыяланган блоктун ичинде (алкагында) саналуучу типтин бардык элементтеринин идентификаторлору константалар катары интерпретацияланышат. Мисалы Spring, Summer, Autumn жана Winter идентификаторлору Season тибиндеги константалар болуп саналат. Бул жерде, бул идентификаторлор жолчо константалар болушпайт. Саналуучу

типтеги идентификаторлор аныкталган бүтүндөй блоктун ичинде бир эле идентификаторду ар түрдүү типтердин ичинде баяндоо ката деп эсептелинет.

Мисалы, төмөнкү программаны трансляциялоодо:

```
program  Duplicate_Identifier;
   type    WeekDay = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
                WorkDay = (Mon, Tue, Wed, Thu, Fri);
      begin
      end.
```

төмөндөгүдөй билдирүү чыгат:

Error: Duplicate identifier (Mon).

Алдын ала аныкталган типтердин константалары саналуучу типтердин маанилери боло алышпайт.

Мисалы, төмөнкү көрүнүштөгү баяндоолор корректтүү эмес.

```
Type
{сандык типтеги константалар}
Digitals = (0,1,2,3,4,5,6,7,8,9 ) ;
{символдук типтеги константалар}
DigSimvols=(‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’) ;
{жолчо тибиндеги константалар}
WeekDay=(‘Mon’, ‘Tue’, ‘Wed’, ‘Thu’, ‘Fri’, ‘Sat’, ‘Sun’);
```

Саналуучу тип иреттик тип болгондуктан, андагы идентификатор-константанын иреттик номери бул типти жарыялоодо идентификаторлордун тизмесиндеги анын позициясы менен аныкталат. Тизмедеги биринчи константа 0 деген, экинчи константа 1 деген ж.у.с. номерлерге ээ болот.

Башка иреттик типтердин өзгөрүлмөлөрү сыяктуу эле саналуучу типтеги өзгөрүлмөлөр үчүн да Ord, Pred жана Succ функцияларын колдонууга болот.

Мисалы,

```
Ord (Summer) = 1;
Pred (Winter) = 2;
Succ (Spring) = 1;
```

Тилекке каршы берилгендердин саналуучу тиби, алардын кеңири колдонулушун чектеп койгон олуттуу жетишпестиктерге ээ. Бул типтеги өзгөрүлмөлөрдүн маанилери Readln оператору менен кийриле жана Writeln оператору менен чыгарыла албайт. Бул жетишпестиктерди жоюу үчүн кошумча кызматчы өзгөрүлмөлөрдү жарыялоого туура келет.

Интервалдык тип

Программада өзгөрүлмөлөрдү баяндоо учурунда, эреже катары, алар кандайдыр бир типтеги маанилердин камтылуучу көптүгүн көрсөтүү үчүн пайдаланылары белгилүү болот. Маанилердин мындай камтылуучу көптүгү Turbo Pascal 7.0 дө берилгендердин интервалдык тиби деп аталуучу типтин жардамында аныкталышы мүмкүн.

Берилгендердин интервалдык тиби мурда аныкталган (программада) типтердин биринин маанилеринин камтылуучу көптүгүн берүү аркылуу аныкталат. Turbo Pascal 7.0 тилинде интервалдык типтеги өзгөрүлмөнүн маанилеринин диапазону, чыныгы типти кошпогондо, каалагандай жөнөкөй типтин жардамында берилет.

Диапазонду берүүдө тиешелүү типтеги өзгөрүлмө кабыл алышы мүмкүн болгон маанилердин эң кичинеси жана эң чоңу (эки константа тең бирдей типте болушу керек) көрсөтүлөт. Мисалы, төмөнкүдөй жарыялоонун жардамында:

```
Type big_bank_notes =0..1000;
```

маанилери бүтүн болгон жана бүтүн сандардын [0..1000] интервалында жаткан берилгендердин жаңы интервалдык тибин түзүүгө болот. Мындай декларация компиляторго бул типтеги өзгөрүлмөлөр үчүн маанилер катары ушул көрсөтүлгөн интервалдагы сандар гана алынышы мүмкүн экендигин көрсөтөт. Программанын аткарылышы учурунда мындай өзгөрүлмөгө бул интервалдан сырткары маанини берүүгө аракет жасоо ката катары каралат жана программанын ишинин токтошуна ошондой эле тиешелүү билдирүүнүн чыгышына алып келет.

Көрсөткүчтүк тип

Turbo Pascal тилинде берилгендердин адрестери жана алардын компьютердин эсинде жайгашышы менен иштөө үчүн арналган каражаттардын тобу бар. Бул жерде биз көрсөткүчтүк тип жөнүндө учкай сөз кылып өтөбүз , анткени бул каражаттар кийинки темаларда атайын каралмакчы.

Көрсөткүчтүк типтеги чоңдуктардын маанилери болуп эстин адрестери келет. Стандарттык Pointer көрсөткүчтүк тибинен айырмаланып пайдалануучунун көрсөткүчтүк тиби, базалык тип деп аталган аныk бир типтеги динамикалык өзгөрүлмөлөр үчүн манилердин көптүгүн аныктайт.

Көрсөткүчтүк типтерди баяндоодо баяндолордун жалпы эрежесинен бир четтөө бар. Кандайдыр бир MyType тибине болгон көрсөткүч ушул MyType тибинин өзүнүн жарыяланышына чейин

баяндалган болушу мүмкүн. Негизгиси, көрсөткөчтүк тип дагы жана ушул көрсөткүч чоңдуктарын көрсөтүп турган тип да жарыялоонун бир эле бөлүгүндө баяндалган болсо болду.

```
Type
      PtrStack =^ Tstack;
     TStack    = record
                   inf: Real;
                   Link: PtrStack;
                 end;
```

Көрсөткүчтүү типтер менен иштөөдө эч нерсени көрсөтпөгөн көрсөткүчтүн маанисин берүүчү nil стандарттык константасы пайдаланылат.

Структураланган типтер

Turbo Pascal дын стуруктураланган типтерине төмөнкүлөр кирет:

- Тип – массив (Array);
- Тип – көптүк (Set);
- Тип – жазуу (record);
- Файлдык тип (file);
- Объекттик тип (object).

Эгер Turbo Pascal тилинин типтерин анын синтаксиси көз карашында эмес, программалоодо пайдаланылуучу берилгендердин структуралары көз карашында карасак, анда жолчолук типтерди да структураланган типтерге кошуу зарыл. Бирок символдук жолчолордун программалодо кеңири колдонулушун эске алуу менен жолчолук типтер, типтердин атайын категориясына бөлүнгөн.

Стуруктураланган типтер, стандарттык типтеги берилгендер сыяктуу бир гана маанини эмес, көп маанилерди кармап туручу берилгендерди баяндоо үчүн кызмат кылат. Ошондуктан стандарттык типтеги берилгендерге караганда, структураланган типтеги берилгендерди баяндоо жана алар менен иштөө бир кыйла татаалыраак.

Өзгөчө объекттик типти белгилөө зарыл. Бул тип, синтаксистик жазылышы боюнча да, өзгөчө маанилик жүгү (смысловая нагрузка) боюнча да башка структураланган типтерден алда канча татаал. Ошондуктан аны көбүнчө өзүнчө категориядагы тип катары карап жүрүшөт.

Процедуралык тип

Процедуралык типтер процедуралардын, функциялардын жана усулдардын аттарын ыйгарууга мүмкүн болгон өзгөрүлмөлөрдү, ошондой эле мындай өзгөрмөлөрдү жана аттарды параметрлер катары берүүгө мүмкүнчүлүк берет.

Процедуралык типтерди баяндоо процедураларды, функцияларды жана усулдарды баяндоо сыяктуу эле синтаксиске ээ болот. Мындай типтердин айырмалап турган өзгөчөлүгү - бул процедуранын/функциянын идентификаторунун жоктугу, анткени тип үчүн процедуранын конкреттүү аты маанига ээ эмес. Андан сырткары, процедуралык типтеги өзгөрүлмөлөргө ыйгарылуучу процедуралар/функциялар, алыскы чакыруунун (дальний вызов) {$F+} директивасы болгон кезде компиляцияланган болушу керек.

Процедуралык типтеги өзгөрүлмө процедура жайгашкан эстин адреси болуп, иш жүзүндө өзгөчө көрүнүштөгү көрсөткүчтүк өзгөрүлмө болуп эсептелет.

Процедуралык типтерди баяндоого жана аларды пайдаланууга карата төмөнкүдөй мисалды келтиребиз.

```
type
 TProc1= procedure (var X,Y:Real);
 TFunc1= function (X:Integer):Real;
 TProc2= procedure(const A:MyArray: var Res: Real);
 TFunc2= function(I,J:Byte;X,Y:Char):string;
  program  ProcTypesDemo;
  type
        TMyFunc=function(X:Integer):Real;
  {$F+}
         function  FirstFunc(X:Integer):Real;
begin
        FirstFunc:=SQR(X)/2;
  end;
  function  SecondFunc(X:Integer):Real;
    begin
       SecondFunc: = SQRT(X)*2;
    end;
   {$F-}
 var       MyFunc: TMyFunc;
 begin
       MyFunc := FirstFunc;
       Writeln (MyFunc(16):7:2); {Жыйынтык - 128 саны}
       MyFunc := SecondFunc;
       Writeln (MyFunc(16):7:2); {Жыйынтык - 8 саны}
  end.
```

3.3.3. Константаларды баяндоо

Константа – бул анык бир типтеги өзгөрбөс чоңдукту (берилгендердин маанилерин) билдирген идентификатор болуп саналат. Константалар дагы өзгөрүлмөлөр сыяктуу эле эсептөө процесинде колдонулуш моментине чейин программанын (же процедуранын) декларациялык бөлүгүндө жарыяланган болушу керек.

Жөнөкөй константалар

Константаларды жарыялоо Const резерврделген сөзү менен башталат. Андан кийин константанын аты, барабардык (=) белгиси жана бул константанын мааниси жазылат.

Мисалы,

```
Const
   {Сандык константалар}
   Length = 100;
   MinNeg = -1 ;   MaxNeg = - 32679 ;   Numb = 7.87e-3;
   {Бульдук константалар}
   Bool1 = True ;    Bool2 = False;
   {Символдук константалар}
   Char8 = ′8′ ;   CharCR = #13;
   {Жолчо константалар}
    Str = ′Borland′ ;       Str2 =  ′Pascal′;
```

Константа жарыялангандан кийин аны жарыяланган аты боюнча программанын каалаган жеринде туруп чакырууга болот.

Turbo Pascal жөнөкөй константалардан сырткары маанилери компиляция убагында эсептеле турган константалык туюнтмаларды колдонууга мүмкүнчүлүк берет. Эгерде төмөндө келтирилген константалар жогорку константалардан кийин жайгашкан деп эсептесек, анда төмөнкү жарыялоолор толук негиздүү болот.

```
Const
        ChrLength = Chr (Length);
        Mean = (MaxNeg - MinNeg) div 2;
        BoolAnd = Bool1 and Bool2;
        CodeOfChar8 = Ord(Char8);
        Name = Str1+Str2+CharCR;
```

Константалык туюнтма так кадимки туюнтмалар сыяктуу эле эрежелердин негизинде баяндалат. Бирок константалык туюнтмаларда колдонууга мүмкүн болгон стандарттык функциялардын тобу төмөнкү функциялар менен чектелген.

Abs, Chr, Hi, Length, Lo, Odd, Ord, Pred, Ptr, Round, Sizeof, Succ, Swap, Trunc.

Типтештирилген константалар

Жөнөкөй константалардан айырмаланып типтештирилген константалардын жарыяланышында анын мааниси менен катар тиби да көрсөтүлөт, б.а. ал, өзгөрүлмөлөрдү жарыялап анан ушул эле жерде ага баштапкы маанини ыйгарууга мүмкүнчүлүк берет.

Мисалы,

```
Const
    MIXER:  Integer = 2000;
    TUK_TUK: Char = #7;
```

Иш жүзүндө типтештирилген константалар эстин статикалык классына ээ болгон өзгөрүлмөлөр болушат. Башкача айтканда алар үчүн баяндалган маанини программанын аткарылышынын башталышында бир гана жолу алып, алар жарыяланган процедурага (функцияга) кайрадан киргенде, кайрадан жаңыдан ысымдаштырылбастан процедуранын (функциянын) мурдагы чакырылышында алган маанисин сактап калат. Типтештирилген константаларды ушундай типтеги өзгөрүлмөлөрдүн дал өзүндөй пайдаланууга болот, айрым учур катары, алар ыйгаруу операторунун сол жагында пайдаланылышы мүмкүн.

Кадимки константалык туюнтмалардан сырткары, типтештирилген константалардын маанилерин берүү үчүн константалык адрестик туюнтмаларды пайдаланышат. ***Константалык адрестик туюнтма*** - бул мааниси глобалдык өзгөрүлмөнүн, типтештирилген константанын, процедуранын же функциянын адреси болгон туюнтма болуп эсептелет. Константалык адрестик туюнтма процедуранын локалдык өзгөрүлмөлөрүнө же динамикалык өзгөрүлмөлөргө кайрыла албайт, анткени компиляция кезинде алардын адрестерин эсептөөгө болбойт.

Типтештирилген константа иш жүзүндө ысымдалуучу өзгөрүлмө болгондуктан ал башка константаларды же типтерди баяндоодо пайдаланылышы мүмкүн эмес.

Turbo Pascal тилиндеги типтештирилген константалардын түрлөрүн карайлы.

Стандарттык типтеги типтештирилген константалар

Типтештирилген константаларга маанилер кадимки барабарлоонун жардамында берилет. Стандарттык типтеги типтештирилген константалардын маанилери болуп стандарттык типтеги берилгендер келет.

Мисалы,

```
const
    Arr_Length:   Integer = 100;
    Step:  Real = 0.001;
    Flag: Boolean = False;
    LineFeeg: Char = #10;
    NewLine: String[2] = #13#10;
     Name :String [14]  = 'Borland Pascal'
var
    Buffer : array [0..1023] of Byte;
const
    BufferOfs : Word = Ofs (Buffer);
    BufferSeg :Word  = Seg(Buffer);
    Ptr  :  Pointer =  @Buffer;
```

Көрсөткүчтүк типтеги типтештирилген константалар

Бул учурда типтештирилген константаларга маанилар катары көрсөткүчтүк типтеги берилгендер берилет.

Мисалы,

```
type
    Ptr = ^Integer;
  const
    IntPtr : Ptr = nil;
    Int1 : Integer = 0;
       Int1Ptr : Ptr = @Int1;
```

Структураланган типтердеги типтештирилген константалар

Turbo Pascal төмөнкү структураланган типтердеги типтештирилген константалар менен иш жүргүзөт:

- "массив" тибиндеги (array);
- "көптүк" тибиндеги (set);
- "жазуу" тибиндеги (record);
- "объекттик" типтеги (ogject).

Структураланган константаны баяндоодо анын ар бир компонентинин мааниси аныk бир синтаксистик эрежелер боюнча көрсөтүлөт.

"Массив" тибиндеги типтештирилген константалар

"Массив" тибиндеги типтештирилген константаларды баяндоодо массивдин ар бир өлчөмүнүн компоненттери өзүнчө кашаага алынат жана үтүр менен ажыратылат. Эң ички кашааларда жайгашкан компоненттер массивдин акыркы өлчөмүнө (эң оң жактагы) тиешелеш келет.

“Массив” тибиндеги константаталардын **мисалдары:**

- бир өлчөмдүү сандык массив

```
const
   DigVector : array [1…7]  of  Real =
            (0.4 ,9.25,  24.32,  55,  41.99,  88.3,  5.4);
```

- эки өлчөмдүү сандык массив

```
const
      DigMatrix : array  [1 .. 3, 1 .. 4]  of  Integer =
          ( (4, 3, 2, 1),   (5, 4, 3, 2),   (6, 5, 4, 3) );
```

Натыйжада төмөнкү көрүнүштөгү матрица түзүлөт

```
4   3   2   1
5   4   3   1
6   5   4   3
```

- үч өлчөмдүү сандык массив

```
const
        Dig3D  :   array  [1 . . 4, 1 . . 3, 1 . . 2]  of  Byte =
        ( ( (1, 2), (1, 2), (1, 2) ),   ( (1, 2), (1, 2), (1, 2) ),
        ( (1, 2), (1, 2), (1, 2) ),  ( (1, 2),(1, 2),(1, 2) ) );
```

- бир өлчөмдүү символдук массив

```
const
CharVect1 :  array [1 . . 6]  of  Char
          =( ′P′ ,  ′A′ ,  ′S′ ,  ′C′ ,  ′A′ ,  ′L′ );
```

же бир кыйла кыска түрдө

```
CharVect2 : array [ 1 . . 6] of  Char = ′PASCAL′;
```

“Көптүк” тибиндеги типтештирилген константа

“Көптүк” тибиндеги канстантанын ар бир компоненти же тиешелүү типтеги жеке константа, же .. (эки чекит) символу менен ажыратылган эки константадан турган маанилердин интервалы болушу мүмкүн.

Мисалы,

```
type
     Digits =  set of    0 . . 9;
     CharDig = set of  ′0′ . .′9′;
 const
     Digset1          :  Digits = [0, 2, 4, 6, 8];
     DigSet2          :  Digits = [1. . 3,  5. .7] ;
     CharDigSet1 :   CharDig = [′0′, ′2′, ′4′, ′6′, ′8′];
     CharDigSet2 :   CharDig = [′0′. .′3′,  ′5′. .′7′] ;
     CharSet        :   Set of Char = [′a′. .′z′,  ′A′. .′Z′];
```

“Жазуу” тибиндеги типтештирилген константалар

Мындай типтеги константаларды баяндоодо жазуунун бардык талааларынын маанилери жана алардын идентификаторлору синтаксистик эреже боюнча көрсөтүлөт.

“Жазуу” тибиндеги типтештирилген константаларда файлдык типтеги талааларга уруксат берилбейт. Варианттык константа-жазууларда талаа-белгинин алдын ала түзүлгөн константасына тиешелүү болгон талаалардын вариантын гана көрсөтүгө уруксат. Талаалар, алар типтин баяндоосунда кандай тартипте келсе ошондой эле тартипте көрсөтүлөт.

“Жазуу” тибиндеги типтештирилген константалардын **мисалдары.**

```
type
        Rec= record
          R : real;
          B : Boolean;
          C : Char;
         end;
         ArrayOfRec = array [1. .3]  of   Rec;
         RecOfArray = record
            ArrInt : array [1. .3]  of  Integer;
            ArrChar : array [1. .2]  of  Char;
         end;
         RecOfRec =  record
                I :  Integer;
               S : String;
               Z : Rec;
        end;
   const
      RecElem : Rec (R : 3.1415; B: Ttue;  C:  '*');
      ArrRec : ArrayOfRec  =  ( ( R : 3.1415 ;  B :  True ;  C:' * '),
                    (R : 0. 0 ;  B :  False ;  C :  ' $ ' ),
                    (R : 6.2832 ;  B : True ;  C ;  ' & ' ) );
Rec Arr: RecOfArray = (ArrInt:  (1, 2, 3) ; ArrChar : ( ' 1 ',  ' 2 ') );
    RecRec : RecOfRec =    (I:  32767 ;   S:  'PASCAL';
                        Z: (R: 3.1415 ;   B:  True ;  C:  ' * ') );
```

Объекттик типтеги типтештирилген константалар

Объекттик типтеги константалардын баяндалышы “жазуу” тибиндеги константаларга окшош аткарылат. Объекттин талааларынын маанилери так ошол сыяктуу көрсөтүлөт, ал эми усулдар (методы) үчүн маанилер көрсөтүлбөгөн болушу да мүмкүн.

Мисалы,

```
type
        Point = object
       X,Y :  Integer;
    end ;
    Rect = object
       A, B :  Point;
    procedure  Init (XA, YA, XB, YB :  Integer);
    procedure  Copy (var R:  Rect);
    procedure  Nove (DX, DY :  Integer);
  end;
const
     ZeroPoint:  Point = (X : 0 ;   Y : 0 );
     ScreenRect:  Rect = (A:  (X : 0 ;   Y : 0);
     B: (X : 80 ;  Y : 25) );
```

Виртуалдык усулдарды кармаган объекттик типтеги константаларды инициализациялоо автоматтык түрдө компиляция этабында аткарылат.

Процедуралык типтеги типтештирилген константалар

Мындай типтеги константа процедуранын же функциянын ыйгаруу боюнча константанын тиби менен биргелешкен идентификаторун көрсөтүшү керек.

Мисалы,

```
type
      FuncType = Function  ( X :   Real );
      Function SH (X :  Real ) :  Real ;  far;
begin
     SH: = (Exp(X)-Exp(-X)) / 2;
end;
  Function CH (X: Real ) :  Real ; far;
  begin
     CH: = (Exp(X)+Exp(-X)) / 2;
  end;
  const
      CurrentFunc: FuncType = SH;
```

3.3.4. Өзгөрүлмөлөрдү баяндоо

Өзгөрүлмө деп программанын ичинде берилгендерди сактоо, коррекциялоо жана берүү (передача) үчүн арналган программанын элементин аташат. Программаларда өзгөрүлмөлөрдү жарыялоо бөлүгү var резерв делген сөзү менен башталат. Өзгөрүлмөнү баяндоо (же жарыялоо) үчүн өзгөрүлмөнүн атын жана анын тибин көрсөтүү зарыл.

Мисалы,

```
var
   A1,  B1 :   Integer;
   B,   C  :   Real;
  D1, K1, F,G : Char;
```

Бул жерде көрүнүп тургандай, бирдей типтеги өзгөрүлмөлөрдү алардын тибин көрсөтүүнүн алдында үтүр менен ажыратып катар жазууга болот.

Өзгөрүлмөлөрдү баяндоодо типтер катары жаңы (программист тарабынан аныкталган) пайдалануучулук типтер, ошондой эле мурда жарыяланып коюлган типтер пайдаланылышы мүмкүн.

Мисалы,

```
type
   Colors= (Red, Blue, Green);
   Vector = array [1 . . 100] of  Integer;
var
 A, B, C  : Real;
  i, j          :  Integer;
  Flag        : Boolean;
 Color      : Colors;
  Digit       : 0 . . 9;
 Season     : (Spring, Summer, Autumn, Winter);
     Vect1, Vect2:   Vector;
    Matrix    : array [1. .5,  1. .10] of  Byte;
```

Бардык өзгөрүлмөлөр глобалдык (процедура жана фукциялардан сырткары жарыяланган) жана локалдык (процедура жана функциялардын ичинде жарыяланган) өзгөрүлмөлөр деп бөлүнөт. Turbo Pascal тилиндеги программаларда өзгөрүлмөлөрдү пайдалануга бир топ чектөөлөр коюлган:

1. глаболдык өзгөрүлмөлөрдүн ичинде бирдей идентификаторлуу эки өзгөрүлмө болбошу керек;
2. локалдык өзгөрүлмөлөр үчүн мындай чектөө бир процедуранын же функциянын пределинде(аймагында) коюлат;

3. программанын текстинде өзгөрүлмөлөр үчүн глобалдык жана локалдык идентификатор бирин-бири кайталай берсе болот, анткени алар эстин ар түрдүү жеринде бирдей атка ээ болгон учурда да бири-биринин маанисине таасир эте албайт.

Absolute директивасынын жардамында, эсте так көрсөтүлгөн адрес боюнча жайгашуучу абсолюттык өзгөрүлмөлөр деп аталган өзгөрүлмөлөр баяндалат. Ар бир абсолюттук өзгөрүлмө өзүнчө баяндалышы керек, б.а. жарыялоодо кош чекиттен мурда турган идентификаторлордун тизмеси бир гана идентификатордон турушу керек.

Абсолюттук өзгөрүлмөлөрдү баяндоонун эки формасы бар:

- өзгөрүлмөнүн жайгашышынын так адресин көрсөтүү формасы;
- бир адрес боюнча эки өзгөрүлмөнү жайгаштыруу формасы.

Биринчи формасы болгон учурда баяндалуучу өзгөрүлмө барып жайгаша турган сегменттин жана жылыштын (б.а. толук адрес) так маанилери көрсөтүлөт.

Мисалы,

```
CrtMode: Byte absolute $0040  $0049;
```

Бул жердеги биринчи константа сегменттин маанисин, ал эми экинчиси ушул сегменттин ичиндеги жылыштын маанисин аныктайт. Эки константа тең $0000 дөн $FFFF (0 дөн 65535 ке) чейинки диапазондун пределинен чыгып кетпеши керек.

Экинчи форма болгон учурда absolute директивасы башка өзгөрүлмөнүн «үстүнө» барып б.а. ошол эле адрестин өзү боюнча барып жайгаша турган өзгөрүлмөнү баяндоо үчүн пайдаланылат.

Төмөнкү мисалдагы жарыялоолордо Digits өзгөрүлмөсү дагы ChDigits өзгөрүлмөсү жайгашкан адреске барып жайгашат деп көрсөтүлгөн. Натыйжада символдордун жолчосун Byte тибиндеги массив менен бастыруу аркылуу массивдин элементерине бул жолчону түзүп турган символдордун коддору ыйгарылат.

```
program TestAbsolute;
 const
    ChDigits : string [10] =  ' 0123456789';
 var  Digits : array [0. .10] of  Byte  absolute  ChDigits;
       i: integer;
 begin
       Writeln (‘«O»  дөн «9» га чейинки символдордун коддору’);
     for   i :=1  to 10  do  Write  ( Digits[i], ' ' );
    Writehn;
 end.
```

Бул программаны ишке салуу төмөнкүдөй жыйынтыкты берет.

Жыйынтык: «0» дөн «9» га чейинки символдордун коддору 48 49 50 51 52 53 54 55 56 57

3.3.5. Процедураларды жана функцияларды баяндоо

Бул бөлүктө тийешелүү синтаксистик эрежелерди сактоо менен процедураларды, функцияларды, конструкторлорду жана деструкторлорду баяндоо аткарылат. Программалар сыяктуу эле процедуралар (функциялар) да аттарга ээ болушат да бир эле процедураны (функцияны) программанын ар түрдүү жеринде туруп ар түрдүү маанилер менен чакырып пайдалануу мүмкүнчүлүгү ишке ашырылат. Конструктор жана деструкторлор процедуралардын атайын типтери болуп виртуалдык жана динамикалык усулдарды ишке ашыруу үчүн арналган. Turbo Pascal тилинин бул элементтери атайын арналган темаларда кеңири каралат.

3.3.6. Экспортту баяндоо

Эскертүү:

exports *бөлүмчөсүн компиляторлордун BP.EXE. BPC.EXE жана BPW.EXE версиялары аркылуу трансляцияланууучу программаларда гана пайдаланууга болот.*

exports - экспорттолуучу аттарды баяндоо бөлүмчөлөрү программанын же динамикалык байланышуучу библиотеканын (DLL) баяндоо бөлүгүнүн каалаган жеринде көп жолу учурашы мүмкүн. exports сүйлөмүндөгү ар бир жазуу экспорттолуучу процедуранын же функциянын идептификаторун берет. Бул учурда бул процедура же функция, анын атын exports сүйлөмүндө көрсөткөнгө чейин баяндалган болушуна көз салуу керек. Андан сырткары, экспорттолуучу процедуранын же функциянын баяндоосу export (аягында “s” тамгасы жок!) директивасын кармап турушу керек. Экспорттолуучу аттар болуп такталган идентификаторлор келиши мүмкүн.

exports баяндоо бөлүмчөлөрүндө берилген программа же динамикалык байланылуучу библиотека тарабынан экспорттолуучу бардык процедуралар жана функциялар саналып жазылат. Программа exports сүйлөмүн кармап турушу мүмкүн болгону менен практикада

бул жетишерлик сейрек кездешээрин байкоого болот. Эреже катары, ал DLL дерде көрсөтүлөт.

exports сүйлөмдөрү программанын же DLLдин баяндоолорунун сырткы бөлүгүндө гана уруксат берилет. Процедуранын, функциянын жана модулдун баяндоо бөлүктөрүндү алар пайдаланыла албайт.

Экспорттун ар бир жазылышы index кызматчы сөзүн камтып турушу мүмкүн. Ал сөздөн кийин 1 ден 32767 ге чейинки диапазондогу бүтүн сан маани турушу мүмкүн. Бул сан маани экспорттолуучу процедурага же функцияга атайын иреттик маанини тиешелеш коет. Эгерде экспорт жазылышында index баяндоосу жок болсо, анда иреттик маани автоматтык түрдө ыйгарылат.

exports сүйлөмүнүн жазылышы name кызматчы сөзүн кармап турушу мүмкүн болуп, ал сөздөн кийин кандайдыр бир жолчолук константа келет. Бул константа процедурага же функцияга ат болуп берилип, кийин ал ошол ат боюнча экспорттолот. Эгерде экспорттун жазылышында name баяндоосу жок болсо, анда процедура же функция өзүнүн символдору жогорку регистрге өзгөртүп түзүлгөн идентификатору менен экспорттолот.

Андан сырткары, экспорттун жазылышы resident кызматчы сөзүн өз ичине алып турушу мүмкүн. Бул кызматчы сөз көрсөтүлгүн учурда экспорт жөнүндөгү информация, азырынча DLL жүктөлүп турганда, оперативдик эсте кала берет. Бул динамикалык жүктөлүүчү библиотекада камтылуучу программаны аты боюнча издөө убактысын бир кыйла азайтат.

3.4. Операторлор бөлүгү (операторлук блок)

Бул программанын структурасындагы бирден-бир сөзсүз жазылуучу бөлүк болуп эсептелет. Классикалык жөнөкөй программа төмөндөгүдөй көрүнүшкө ээ болот.

```
bedin
      Writeln (′Hello, World′);
  end.
```

Бирок, жогоруда айтылгандай мындан мурдагы программанын биз карап өткөн бөлүктөрүн пайдаланбай туруп эң примитивдүү программаларды гана жазууга болот.

Бул бөлүктө жазууга уруксат берилген операторлордун көптүгү өзүнчө темада каралмакчы.

Контролдук тапшырмалар

1. Кайсы программаларда каталар бар? Ал каталарды көрсөткүлө.

а)
```
program  P1;
    procedure   Sub1;
    begin
    end;
  var
      A,  B  : Real;
    type
      TV  = array  [1. .10] of Integer;
    const
      k=20;
    label   _L1;
    begin
    end.
```

б)
```
program  P2;
  type
    TM  = array  [1. .N] of Real;
  const
    N=100;     M=50;
var
    S,  T  = array  [1. .20]  of   Byte;
begin
end.
```

в)
```
program  P3;
  label
     10;
  const
     M = 200;
  type
    TV  = array  [1. .M] of  Char;
var
    X, M;
begin
end.
```

2. Бир эле убакта төмөнкү бүтүн типтердин ар бирине таандык болгон константалардын мисалдарын жазгыла.

   а) Shortint, Integer;
   б) Longint, Shortint;
   в) Byte, Word;
   г) Word, Shortint;

3. Бир эле убакта төмөнкү чыныгы типтердин ар бирине таандык болгон константалардын мисалдарын жазгыла.

   а) Singl, Comp;
   б) Double, Single;
   в) Real, Extended;
   г) Real, Double, Extended, Comp;
   д) Extended, Comp;

4. Бульдук, символдук, жолчолук жана көрсөткүчтүк типтердеги константалардын мисалдарын жазгыла.
5. Пайдалануучу тарабынан аныкталган типтердин ар биринин баяндоолорунун жана алардын ар бири үчүн константалардын мисалдарын жазгыла.

4. АМАЛДАР ЖАНА ТУЮНТМАЛАР

Билимдүүнүн белгиси, билгенин келет бергиси,
Акылдуунун белгиси, акылын келет бергиси.
(Даанышман ойлор)

4.1. Туюнтмалар, амалдар жана операнд түшүнүктөрү

Туюнтмалар константалардан, өзгөрүлмөлөрдөн жана алардын комбинацияларынан, амалдардан, кашаалардан жана функциялардан түзүлүшү мүмкүн. Мындагы тиешелүү амалдарды аткаруу менен бул туюнтманын мааниси деп аталган жалгыз маани алынышы керек.

Туюнтмалар амалдардан жана операнддардан турат. Операнддардын санына карата амалдар унардык жана бинардык деп бөлүнөт. Унардык амалдар бир гана операндга ээ болуп, анын алдына амалдын символу жайгашат.

Мисалы,

Туюнтма	Жыйынтык
-7	-7
-(-8)	8
NotFalse	True

Көпчүлүк амалдар бинардык болуп эсептелет. Мындай амалдар эки операндга ээ болуп ал операнддардын ортосуна амал белгиси (символу) коюлат.

Мисалы,

Туюнтма	Жыйынтык
6+8	14
(4-2)*6+12	24
True or False	True

4.2. Амалдардын приоритети жана алардын классификациясы

4.2.1. Амалдардын приоритети

Туюнтмалар менен иштөөдө андагы амалдардын аткарылыш удаалаштыгын так билүү башкы мааниге ээ болот. Андагы амалдардын аткарылыш удаалаштыгы төмөнкүдөй үч фактор менен аныкталат:

- амалдардын приоритети менен;
- туюнтмадагы амалдардын жайгашуу тартиби менен;
- кашааларды пайдалануу менен.

Приоритет боюнча бардык амалдар төрт группага бөлүнөт. Биринчи (жогорку) приоритеттеги амалдар биринчи кезекте аткарылат. Ал эми (төмөнкү) төртүнчү приоритеттеги амалдар акыркы кезекте аткарылат. Приоритеттери барабар болгон амалдар солдон оңду карай аткарылат, бирок кээде компилятор бир кыйла оптималдуу кодду генерациялоо үчүн операндарды башкача иреттеп алышы мүмкүн. Кашаалар амалдарды кадимки аткарылыш тартибин өзгөртүү үчүн кызмат кылышат. Кашаалардын ичине алынган туюнтмалар өзүнчө операнд катары эсептелинип чыгылып, кийин анын жыйынтыгы кашаалардан сырткары турган амалдарды аткаруу үчүн пайдаланылат.

Амалдардын приоритети

Приоритети	**Амалдар**	**Амалдардын категориясы**
Биринчи (жогорку)	+ - not @	Унардык амалдар.
Экинчи	* / div mod and shl shr	Көбөйтүү тибиндеги бинардык амалдар.
Үчүнчү	+ - or xor	Кошуу тибиндеги бинардыкамалдар
Төртүнчү (төмөнкү)	= <> < > < = > = In	Биннардык катыш амалдары

Татаал туюнтмалардын мисалдары:

1. A*(((x-y/z)*3+5)/(b+c)-D*E)-14
2. (A<>B) and ((I<J)or(J<K)) and (C in D)
3. (5-abs(x))/(cos(x)+sin(x))+3*exp(y)+sqr(z)
4. (4.2*sqrt(int(abs(x))))/(pi-arctan(x))-4/3
5. (sqrt(sin(x)-0.0001*cos(sqrt(y))))/(1-x*y)

4.2.2. Амалдарды классификациялоо

Turbo Pascal 7.0 тилинде бардык амалдарды төмөндөгүдөй тайпаларга бөлүүгө болот:

1. Унардык: **+** , **-**
2. Бинардык: **+** , **-** , ***** , **/** , **div** , **mod**
3. Катыш амалдары: **=** , **<>** , **<** , **>** , **<=** , **>=**

4. Бульдук (логикалык) амалдар: not, and, or, xor
5. Разряд боюнча логикалык жана жылыш амалдары:
   not , and , or , xor , shl , shr
6. Жолчолор жана символдор үстүнөн амал (конкатенация): +
7. Көптүктөр үстүнөн болгон амалдар: + , - , * , in , <= , >=
8. Адрести алуу амалы: @

4.3. Амалдарды баяндоо

Биз бул жерде таблицалардын жардамында амалдардын аракети, алардын операнддарынын жана жыйынтыктарынын типтери жөнүндөгү малыматтарды келтирип кетебиз.

4.3.1. Арифметикалык амалдар

+ , _ , * , / амалдарынын жардамында аткарылуучу аракеттердин маңызы өзгөчө түшүндүрүүнү талап кылбайт. Бир гана эсте тутуучу нерсе, эгер операнддар ар түрдүү типте болсо, анда жыйынтык бул типтердин бардыгын өз ичине алчу тип болот, ал эми бөлүү **(/)** амалынын жыйынтыгынын тиби дайыма чыныгы болот.

Div амалынын жыйынтыгы бүтүн сан маани болуп, ал биринчи операндды экинчи операндга бөлгөндөгү жыйынтыктын бүтүн бөлүгүнө барабар болот.

Mod амалынын жыйынтыгы болсо, биринчи операндды экинчи операндга бөлүүнүн калдыгына барабар болгон бүтүн сан мааниге барабар.

Амал белгиси	Амалдын аталышы	Операнддардын типтери	Жыйынтыктын тиби
Унардык амалдар			
+	**Белгини сактоо**	**Бүтүн Чыныгы**	**Бүтүн Чыныгы**
-	**Белгини тануу**	**Бүтүн Чыныгы**	**Бүтүн Чыныгы**
Бинардык амалдар			
+	**Кошуу**	**Бүтүн Чыныгы**	**Бүтүн Чыныгы**
-	**Кемитүү**	**Бүтүн Чыныгы**	**Бүтүн Чыныгы**
*	**Көбөйтүү**	**Бүтүн Чыныгы**	**Бүтүн Чыныгы**
/	**Бөлүү**	**Бүтүн Чыныгы**	**Чыныгы Чыныгы**
div	**Бүтүн сандык бөлүү**	**Бүтүн**	**Бүтүн**
mod	**Бөлүүдөгү калдык**	**Бүтүн**	**Бүтүн**

Мисалдар:

Туюнтма	Жыйынтыгы
11 div 3	3
11 mod 3	2
36 div 7	5
36 mod 7	1

4.3.2. Катыш амалдары

Катыш амалдарынын аракети алардын математикалык түшүндүрүлүшү менен тиешелеш келип жыйынтыктары бульдук типтеги маанилер (True, False) болот.

Бул амалдарды сандык маанилер үчүн колдонуу анчалык кыйын эмес, бирок аларды жолчолук маанилер үчүн колдонуу эрежесин билүү зарыл. Бул учурда салыштыруу ар бир символ боюнча ASCII нин кеңейтирилген таблицасындагы (3-тиркеме) алардын коддорунун маанилерине жараша оңдон солду карай аткарылат.

Бардык жолчолук маанилер алардын узундуктарынан көз карандысыз түрдө биргелешкен деп эсептелинет. Андан сырткары, символдук типтеги маанилер жолчолук типтеги маанилер менен биргелешкен болуп узундугу бирге барабар болгон жолчо катары эсептелинет.

Көрсөткүчтүк типтеги операндарды салыштырууда = жана <> амалдарын дагы колдонууга уруксат берилет. Эки көрсөткүч, качан алар экөө тең бир жана бир гана объектти көрсөтүп турганда барабар болот.

Амалдар	**Аталышы**	**Операндардын типтери**	**Жыйынтыгы -нын тиби**
=	Барабар	Биргелешкен жөнөкөй, жолчолук же көрсөткүчтүк тип	Бульдук
< >	Барабар эмес	Биргелешкен жөнөкөй, жолчолук же көрсөткүчтүк тип	Бульдук
<	Кичине	Биргелешкен жөнөкөй же жолчолук тип	Бульдук
>	Чоң	Биргелешкен жөнөкөй же жолчолук тип	Бульдук
<=	Кичине же барабар	Биргелешкен жөнөкөй же жолчолук тип	Бульдук
>=	Чоң же барабар	Биргелешкен жөнөкөй же жолчолук тип	Бульдук

Мисалдар:

Туюнтма	Жыйынтыгы	Туюнтма	Жыйынтыгы
(3+4) = (5+6)	False	(8-6) <= -2	False
False < > True	True	(8-5) >= -1	True
'ABC' < 'АВД'	True	'ABC' > 'A'	True

4.3.3. Бульдук (логикалык) амалдар

Бульдук амалдар буль алгебрасынын эрежелерине ылайык аткарылат. Ал эрежелерди төмөнкү таблицада келтиребиз.

Операндар		Амалдар			
A	B	not A	A and B	A or B	A xor B
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False

Turbo Pascal 7.0 тилинде логикалык туюнтмалардагы бульдук амалдарды аткарууда компилятор коддорду генерациялоонун ар түрдүү эки модели менен иш жүргүзөт:

- толук схема боюнча эсептөө;
- кыска схема боюнча эсептөө.

Толук схема боюнча болгондо бардык операндар эсептелет жана туюнтманын мааниси биринчи амалды аткаргандан кийин эле белгилүү болуп калса да андагы бардык амалдар аткарылат.

Кыска схеманы пайдаланууда операндарды жана амалдардын жыйынтыгын эсептөө катуу тартипте солдон оңду карай аткарылып, андан аркы амалдарды аткаруу туюнтманын акыркы жыйынтыгына таасирин тийгизбей турган болуп калары менен эсептөө токтойт.

Эсептөөнүн моделин башкаруу компилятордун {$B} директивасы менен ишке ашырылат. Атайы көрсөтүлбөгөн учурда эсептөөлөрдүн кыска схемасына туура келүүчү {$B-} директивасы пайдаланылат.

Амал белгиси	Амалдын аталышы	Операндардын типтери	Жыйынтыктын тиби
Унардык			
not	Логикалык тануу	Бульдук	Бульдук
Бинардык			
and	Логикалык **ЖАНА**	Бульдук	Бульдук
or	Логикалык **ЖЕ**	Бульдук	Бульдук
xor	**ЖЕ** амалын логикалык төгүндөө	Бульдук	Бульдук

4.3.4. Разряддар боюнча (биттик) бульдук жана жылыш амалдары

Turbo Pascal 7.0 тилиндеги разряддар боюнча болгон амалдар бүтүн типтеги гана операнддарга ээ боло алат. Бул амалдар операнддардын экилик көрсөтүлүштөрүнүн үстүнөн жүргүзүлүп, разряддар боюнча аткарылат.

Not амалынын жыйынтыгы берилген операнддын разряддары боюнча инвертирленген экилик көрсөтүлүшкө тиешелеш келүүчү ондук санга барабар болгон бүтүн сан маани болот.

And амалынын жыйынтыгы берилген операнддардын үстүнөн разряддар боюнча аткарылган логикалык ЖАНА амалынын жыйынтыгы болуп эсептелген экилик санга тиешелеш келүүчү ондук санга барабар болгон бүтүн сан маани болот.

Or амалынын жыйынтыгы берилген операнддардын үстүнөн разряддар боюнча аткарылган ЖЕ логикалык амалынын жыйынтыгы болуп эсептелген экилик санга тиешелеш келүүчү ондук санга барабар болгон бүтүн сан маани болот.

Xor амалынын жыйынтыгы, берилген операнддардын үстүнөн разряддар боюнча аткарылган ЖЕ ни төгүндөө логикалык амалынын жыйынтыгы болуп эсептелген экилик санга тиешелеш келүүчү ондук санга барабар болгон бүтүн сан маани болот.

A Shl B амалынын жыйынтыгы A операндынын экилик көрсөтүлүшүн разряддар боюнча солго B разрядга жылыштыруунун натыйжасында алынган экилик санга тиешелеш келүүчү ондук санга барабар болгон бүтүн сан маани болот. Мында бошоп калган разряддар нөлдөр менен толукталат.

A Shr B амалынын жыйынтыгы A операндынын экилик көрсөтүлүшүн разряддар боюнча оңго B разрядга жылыштыруунун натыйжасында алынган экилик санга тиешелеш келгүүчү ондук санга барабар болгон бүтүн сан маани болот. Бул жерде да бошоп калган разряддар нөлдөр менен толукталат.

Амалдар	Аталышы	Операндардын типтери	Жыйынтык-тын тиби
Унардык			
Not	Разряддар боюнча тануу	Бүтүн	Бүтүн
Бинардык			
And	Раздяддар боюнча ЖАНА	Бүтүн	Бүтүн
Or	Разряддар боюнча ЖЕ	Бүтүн	Бүтүн
Xor	Разряддар боюнча ЖЕ ни төгүндө	Бүтүн	Бүтүн
She	Разряддар боюнча солго жылыш	Бүтүн	Бүтүн
Shr	Разряддар боюнча оңго жылыш	Бүтүн	Бүтүн

Мисалдар: Эгерде A жана B операндары Byte тибине ээ болсо, анда A=11, B=2 болгондо жогоруда каралган амалдар төмөнкүдөй жыйынтыктарды берет.

Операндар үстүнөн болгон амалдар	**Ондук маани**	**Экилик көрсөтүлүш**
Операнд A	11	00001011
Операнд B	2	00000010
not A	224	11110100
A and B	2	00000010
A or B	11	00001011
A xor B	9	00001001
A shl B	44	00101100
A shr B	2	00000010

4.3.5. Жолчолук амал

Эки жолчону же символду конкатенациялоо (уламалоо) амалын белгилөө үчүн Turbo Pascal да кадимки кошуу (+) амалы сыяктуу символ пайдаланылат. Бул амалдын жыйынтыгы болуп биринчи операндын символдорунун акырына экинчи операндын символдорун улап жазуу менен түзүлгөн эки операндын тең символдорунан турган жолчо эсептелет. Эгер жыйынтык-жолчонун узундугу 255 тен ашып кетсе, анда биринчи 255 символ калтырылып калганы кесилип ташталат.

Амал	**Аталышы**	**Операндар тиби**	**Жыйынтыгынын тиби**
+	Конкатенация (уламалоо)	Жолчолук, символдук	Жолчолук

Мисалдар:

Туюнтма	**Жыйынтык**
′turbo′+′Pascal′+′7.0′	′TurboPascal 7.0′
′Асан′+′бай′	′Асанбай′
′АВС′+′Д′	′АВСД′
′X′+′..′+′Z′	′X..Z′
′May′+′ Type′	′MayType′

4.3.6. Көптүктөр үстүнөн амалдар

Көптүктөр үстүнөн болгон амалдар көптүктөр теориясынын эрежелери боюнча аткарылат.

Эгер A жана B көптүктөр болсо, анда

- бул эки көптүктүн биригүүсү болгон A+B амалынын жыйынтыгы A көптүгүнүн да ошондой эле B көптүгүнүн да бардык элементтерин өзүнө камтыган C көптүгү болот.
- бул эки көптүктүн айырмасы болгон A-B амалынын жыйынтыгы A көптүгүнүн B көптүгүнө кирбеген элементтеринен гана түзүлгөн C көптүгү болот.
- бул эки көптүктүн кесилишүүсү болгон A*B амалынын жыйынтыгы бир мезгилде A жана B көптүктөрүнүн экөөнө тең таандык болгон элементтерден түзүлгөн C көптүгү болот.

Эгер көптүктүн үстүнөн болгон амалдын жыйынтыгынын эң кичине мааниси X болсо, ал эми эң чоңу У болсо, анда жыйынтыктын тиби Set of X..Y болот.

A=B салыштыруу амалынын мааниси True, ал эми A<>B салыштыруу амалынын мааниси False болот ошондо жана ошондо гана качан А жана В көптүктөрү бирдей элементтерди кармап турса.

A<=B салыштыруу амалынын жыйынтыгы True болот, эгер A көптүгү B көптүгүнүн камтылуучу көптүгү болсо.

A>=B салыштыруу амалынын жыйынтыгы True болот, эгер A көптүгү B көптүгүнүн элементтерин өз ичине камтып турса.

x in A таандык болуу амалынын жыйынтыгы True болот, эгер кандайдыр бир T иреттик тибиндеги x мааниси, ушул эле T тибиндеги A көптүгүнүн элементи болсо.

Мисалдар:

Туюнтма	**Жыйынтыгы**
[1,2,3,4]+[3,4,5,6]	[1,2,3,4,5,6]
[3,5,7,9]-[1,3,6,7]	[5,9]
[1,2,3,4]*[3,4,5,6]	[3,4]
[1,2,3]=[1,2,3,4]	False
[1,2,3]<>[1,2,3,4]	True
[1,2,3]<=[1,2,3,4]	True
[1,2,3]>=[1,2,3,4]	False
4 in [3,4,5,6]	True

Көптүктөр үстүнөн болгон амалдарды аткарууда операндар үчүн уруксат берилген типтерди жана алынган жыйынтыктардын типтерин төмөнкү таблицада көрсөтөбүз.

Амалдар	Аталышы	Операнддардын типтери	Жыйынтыктын тиби
+	Биригүү	Көптүктөрдүн биргелешкен типтери	Көптүк
-	Айырма		Көптүк
*	Кесилишүү		Көптүк
=	Барабардык		Бульдук
<>	Барабар эместик		Бульдук
>=	Камтуучу көптүк болот		Бульдук
<=	Камтылуучу көптүк болот		Бульдук
in	Таандык болуу	Сол операнд: каалагандай Т иреттик тиби. Оң операнд: Т тибиндеги көптүк.	Бульдук

4.3.7. Адрести алуу (көрсөткүчтү алуу) амалы

@ амалы унардык операция болуп бул амалды аткаруунун жыйынтыгы катары анын операндына болгон көрсөткүч эсептелет. Жыйынтыктын тиби nil көрсөткүчүнүн тиби менен биргелешкен болот, бул болсо жыйынтыкты каалагандай көрсөткүчтүк өзгөрүлмөгө ыйгарууга мүмкүнчүлүк берет.

@ амалынын операнды катары өзгөрүлмөлөрдүн, процедуралардын, функциялардын жана усулдардын (методдордун) идентификаторлору пайдаланылышы мүмкүн.

Амал	аталышы	Операнддын тиби	Жыйынтыктын тиби
@	Көрсөткүчтү алуу	Өзгөрүлмөнүн, процедуранын, функциянын же усулдун идентификатору	nil менен биргелешкен көрсөткүч

Тапшырмалар

1. Төмөнкү формулаларды Turbo Pascal тилинде жазгыла:

а) $\left(\sqrt{e^{(x+y)}} + X^{|y|} + Z\right)$

б) $\frac{Y^{x+1}}{\sqrt{|Y-2|+3}} + \frac{X+\frac{Y}{2}}{2|X+Y|}$

в) $(X+1)e^{y+3} + \frac{\sqrt{X-1} - \ln(|Y|)}{1+\frac{X^2}{2}+\frac{X^3}{3}} > \frac{A-B}{1-\frac{1}{X}}$

2. Төмөнкүдөй баяндоолор берилген:

```
var
 i, j : Integer;
 x, y: Real;
 a,b: Boolean;
 c : Char;
 s : String;
```

Анда төмөнкү операторлордун кайсыларында каталар бар, болсо эмне үчүн ката?

1) x := i*y ;
2) j := i*y ;
3) i := j/2 ;
4) i := j div 2 ;
5) x := y mod 2 ;
6) i := j mod i ;
7) j := i or 100 ;
8) x := y and j ;
9) a := not(a and b) ;
10) b := ((x-2) < 2) = a ;
11) c := c+s ; 12) s := s +c ;

5. ОПЕРАТОРЛОР

Насаатты угуп жашаган,
Туура жолдо бараткан.
Катасын айтсаң укпаган,
Туура жолдон адашкан.
(Дауд уулу Сулайман)

Оператор – бул алгоритмдеги анык бир аракеттерди ишке ашырууга мүмкүндүк берүүчү программанын бөлүнбөс элементи болуп эсептелет. Turdo Pascal тилинин бардык операторлорун шартуу түрдө эки тайпага бөлүүгө болот:

- жөнөкөй операторлор;
- структуралык операторлор.

Программада пайдаланылуучу операторлор бири-биринен ошондой эле программалык башка бардык элементтерден (;)-“үтүрлүү чекит” символу менен ажыратылган болушу керек. Бул символ оператордун бир бөлүгү эмес, ал ажыраткыч белги болуп саналат. Ошондуктан программанын акыркы операторунан кийин жана курама оператордун акыркы операторунан кийин, башкача айтканда end кызматчы сөзүнөн мурун үтүрлүү чекитти коюу шарт эмес. Эгерде ушундай учурда бары бир “;” символу коюлуп калса, анда ал оператордон кийин курук (бош) оператор жайлашкан деп саналат жана ката болуп эсептелбейт.

Мисалы,

```
readln (A,B,C);
  a:=b;   b:=c;
    . . .
  95:  Writeln (′A=′A);
begin
    . . .
        Writeln(′C=′C);
end.
```

Бул мисалдан көрүнүп тургандай ар кандай оператордун алдында эн-белги (метка) туруп калышы мүмкүн. Ал goto өтүү операторунун жардамында программанын ушул чекитине башкарууну берүү үчүн пайдаланылат. Ал эми эн-белгилерди (меткаларды) баяндоону биз жогоруда карап өткөнбүз.

5.1. Жөнөкөй операторлор

Жөнөкөй оператор – бул өзүнүн ичинде башка операторлорду кармап турбаган оператор болуп саналат. Turbo Pascal тилинде мындай операторлорго төмөнкүлөр кирет:

- ыйгаруу оператору (:=);
- шартсыз өтүү оператору (goto);
- процедурага (функцияга) кайрылуу оператору.

Буларга өз-өзүнчө токтолуп өтөлү.

5.1.1. Ыйгаруу оператору

Ыйгаруу оператору өзгөрүлмөнүн идентификаторунан же пайдалануучулук функциядан, «:=» - ыйгаруу символунан жана туюнтмадан турат. Сол жагында функциянын идентификатору турган ыйгаруу оператору ошол функциянын телосунун пределинде гана жайгашкан болушу мүмкүн.

Ыйгаруу операторунун аткарылышы туюнтма менен аныкталуучу маанини эсептөөгө жана ал маанини ыйгаруу (**:=**) белгисинин сол жагында турган ат менен идентификацияланууучу өзгөрүлмөгө ыйгарууга алып келет.

Сол жакта функциянын идентификатору туруп калган учурда функциянын маанисин аныктоо, башкача айтканда функция аркылуу чакыруу чекитине кайтарылуучу маанини аныктоо аткарылат.

Жөнөкөй типтеги гана эмес массивдер, жазуулар жана көптүктөр сыяктуу структураланган типтердеги маанилерди да ыйгарууга уруксат берилет.

Сөзсүз талап кылынуучу нерсе,

Ыйгаруу операторунун оң жагында турган туюнтманын тиби оператордун сол жагындагы өзгөрүлмөнүн же функциянын жыйынтыгынын тиби менен ыйгаруу боюнча биргелешкен болушу керек.

Бул эрежеден бир гана четтөө бар: ыйгаруу операторунун сол жагында чыныгы типтеги өзгөрүлмө, ошол эле учурда анын оң жагында бүтүн типтеги туюнтма туруп калышы мүмкүн.

Мисалдар:

```
Var
    A, B, C  : Real;
    i, j, k     : integer;
    Flag     : Boolean;
```

```
    Vec1, Vec2 : array [1..10] of Byte;
     Rec1, Rec2:   record
                      Ch   :  char;
                      X    :  Real;
                   end;
  Set1, Set2 :  set   of  Char;
    Begin
          .  .  .
        A := B*C;
        i  :=  j  div  k;
        Flag  :=  (i<>1) and  (B<C);
        Vek1:= Vek2;
        Rec1:= Rec2;
        Set1:= Set2;
    end.
```

5.1.2. Шартсыз өтүү оператору

Шартсыз өтүү оператору goto кызматчы сөзүнөн жана андан кийин көрсөтүлүүчү эн-белгиден (меткадан) турат.

goto операторунун аткарылышы ушул goto операторунда көрсөтүлгөн эн-белги (метка) менен белгиленген операторго башкарууну берүүгө алып келет.

Бул жерде сөзсүз талап кылынуучу нерсе, шартсыз өтүү операторунда көрсөтүлгөн эн-белги (метка) goto операторунун өзү жайгашкан блокто же модулда жайгашкан болушу керек. Бул болсо өтүү операторунун жардамында процедуранын (функциянын) ичине башкарууну берүүгө ошондой эле башкарууну процедурадан (функциядан) аны курчап турган чөйрөгө берүүгө тыюу салынарын билдирет.

Дагы белгилеп кетүүчү нерсе, goto оператору структуралык программалоонун принциптерине каршы келет, ошондон улам аны программаларды жазууда колдонуу сунуш кылынбайт. Андан сырткары, Turbo Pascal 7.0 тилинде алдын ала аныкталган Break жана Continue процедуралары бар болуп алар goto оператору көбүрөөк колдонула турган ситуациялар үчүн атайын арналган.

Мисал.

```
 program Examp_Goto_Exit;
   uses CRT;
     {Программага Crt библиотекасын кошобуз, себеби анда }
     {KeyPressed процедурсынын баяндамасы бар}
    Label m1, m2 ; { m1, m2  глобалдык меткаларын жарыялоо}
```

```
Var N: integer;
Begin
   N:=0;
 m1:
WriteIn ('1-меткага өтүүлөрдүн саны =', N);
  If  KeyPressed then exit;
        {Эгер каалагандай клавиша басылса (KeyPressed),}
        {программанын ишин аяктоо керек}
  If  N>200  then goto  m2;
        {Эгер N өзгөрүлмөсүнүн мааниси 200 дөн чоң болсо,}
        { анда  m2  меткасынан кийин турган операторго өт }
  N:=N+1;
  goto m1; {m1 меткасынан кийин турган операторго өт}
  m2:
  for N:=1to  100 do
  begin
  if KeyPressed then break;
        {Эгер каалагандай клавиша басылса, (Keg Pressed),}
        { анда  for  to  do  цикилинен чык}
  if  N <> round (N/10)*10   then   continue;
        {Эгер N өзгөрүлмөсү 10 го калдыксыз бөлүнбөсө,}
          { анда for  циклинин жаңы итерациясына өт}
WriteIn  (′2- меткага өтүүлөрдүн саны =′,N);
end;
halt (1); {1 деген аяктоо коду менен программа ишин аяктайт}
end.
```

Келтирилген программа практикада колдонула албайт, бирок шартсыз өтүү операторунун мүмкүнчүлүктөрүн жана функцияларын көрсөтмөлүү демонстрациялай алат. Бул программаны талдоо үчүн циклдер жөнүндө жакшы түшүнүк болуш керек эле ошондуктан цикилдик операторлор менен танышкандан кийин бул мисалга кайра кайрылып талдап чыгууну сунуш кылабыз.

5.1.3. Процедурага (функцияга) кайрылуу оператору

Процедурага кайрылуу оператору идентификатордон жана анын соңунан түздөн-түз тегерек каашалардын ичинде жайгашылуучу иш жүзүндөгү (фактические) параметрлердин тизмесинен турат. Параметрлери жок процедурага кайрылуу оператору процедуранын идентификаторунан гана турат.

Процедурага кайрылуу операторунун аткарылышы процедуранын телосунда баяндалган аракеттерди активдештирүүгө

алып келет. Операторду көрсөтүлгөн иш жүзүндөгү параметрлер процедуранын бөркүндө жайгашкан формалдык параметрлер менен типтери боюнча, саны жана өз ара жайгашыштары боюнча тиешелеш болушу керек. Процедуранын телосун активдештирүүнүн алдында иш жүзүндөгү параметрлерди тиешелүү формалдык параметрлерге берүү процесси болуп өтөт.

Ушуга эле окшош аракеттер объекттик типтерде баяндалган усулдарды (методдорду) чакырууда аткарылат.

Мисалы,

```
Summa  (A, m, n, sum);
        Initialize;
        Swap(x, y);
```

5.2. Структуралык операторлор

Структуралык операторлор өзүнүн ичине башка операторлорду кармап турат жана алардын аткарылыш удаалаштыгын башкарат.

Turbo Pascal тилинде структуралык операторлор болуп төмөнкүлөр эсептелет:

1) курама оператор;
2) шарттуу операторлор:
   - альтернатива оператору if;
   - тандоо оператору case;
3) цикл операторлору:
   - алдын ала шарттуу цикл оператору while;
   - шарты аягында келген цикл оператору repeat;
   - эсептегичтүү цикл оператору for;
4) жазуулар үчүн with оператору.

5.2.1. Курама оператор

Курама оператор операторлордун кандайдыр бир тайпасын бир бүтүнгө бириктирип, андан соң алар бир оператор катары каралып калат.

Курама оператор операторлук кашаалар деп аталган begin жана end кызматчы сөздөрүнүн ортосунда жайгашкан бириктирилүүчү операторлордун удаалаштыгынан турат.

Turbo Pascal дын синтаксиси программанын кандайдыр бир чекитинде бир гана операторду көрсөтүүгө уруксат берсе, ал эми алгоритм боюнча бул жерде операторлордун тайпасын аткаруу зарыл болуп калса мындай учурларда, эреже катары, курама оператор

пайдаланылат. Андай оператор, эреже катары, структуралык операторлор менен биргелешип пайдаланылат.

Мисалы,

```
while   A<>0 do
        begin
          writeln ('Санды кийригиле');
           readln (A);
           writeln (A, 'санынын квадраты=', A*A);
        end;
```

5.2.2. Шарттуу операторлор

Turbo Pascal тилинде эки if жана case шарттуу өтүү операторлору бар болуп алардын ар бири эки – толук жана толук эмес формада жазылышы мүмкүн. Ошентип бул төрт форма алгоритмдердин төрт жалпыланган башкаруу констпукцияларына тиешелеш келет.

if жана case операторлору маңызы боюнча экөө тең шарттуу өтүү операторлору болсо да, тарыхый түрдө if оператору шарттуу өтүү оператору деп, ал эми case оператору тандоо же вариант оператору деп аталып калган.

<u>if шарттуу оператору</u>

Биз жогоруда белгилегендей if шарттуу оператору толук жана толук эмес формаларда жазылган болушу мүмкүн. Ал формалар төмөнкүдөй көрүнүшкө ээ.

Толук эмес формасы:

```
If туюнтма
   then оператор
```

Толук формасы:

```
If туюнтма
   then оператор1
   else оператор2
```

Шарттуу оператордун аткарылышында адегенде туюнтма эсептелет, анын жыйынтыгы бульдук типти гана кабыл ала алат, андан кийин жыйынтыктын маанисинен көз кранды түрдө же then кызматчы сөзүнөн кийин турган Оператор1 аткарылат (жыйынтык true болсо) же else кызматчы сөзүнөн кийин турган Оператор2 (жыйынтык false болсо) аткарылат.

If оператору толук эмес формада жазылган учурда жыйынтыктын мааниси false болсо, анда башкаруу if операторунан

кийин турган операторго түздөн-түз берилет, ал эми then кызматчы сөзүнөн кийин турган оператор каралбай калып кетет.

If операторун колдонууда анын төмөнкүдөй синтаксистик өзгөчөлүктөрүнө көңүл буру керек.

Биз билгендей Turbo Pascal тилинде операторлор бири-биринен "**;**" символу менен ажыратылат. Бул болсо структуралык операторду ичинде "**;**" символу кездешпестиги керек дегендикти билдирет. Андай болбогондо ал символдон кийин турган ар кандай нерсе башка оператор катары эсептелинет.

Катасы бар мисал.

```
if A>B
   then C:= A;   ← Ката!
   else C:= B;
```

Бул көрсөтүлгөн мисалдагы else кызматчы сөзүнөн мурун турган үтүрлүү чекит if операторунун текстин аяктайт дагы ал болсо синтаксисттик катага алып келет, себеби Turbo Pascal тилинде else кызматчы сөзү менен башталган оператор жок. Мына ушундай катаны кетирбес үчүн төмөнкү эрежени эсте тутуу зарыл.

else кызматчы сөзүнөн мурун эч качан «;» - үтүрлүү чекит символу коюлбайт.

Синтаксиси боюнча then жана else кызматчы сөздөрүнөн кийин бир гана оператор турушу мүмкүн.

Эгерде альтернативанын бутактарынын бирөөсүндө же экөөндө тең бир нече операторлорду аткаруу талап кылынса, анда операторлордун тайпасын бир оператор катары интерпретациялоого мүмкүнчүлүк берүүчү

```
begin
…
end;
```

курама операторун пайдаланууга болот.

Ушул begin жана end кызматчы сөздөрүнүн ортосунда жайгашкан операторлор бири-биринен, башка бардык жердегидей эле, үтүрлүү чекит менен ажыратылган болушу керек.

If операторунун жөнөкөй учурунун жалпыланган формалары төмөнкү жадыбалда келтирилген.

Бутактагы оператор-лордун саны		**if операторунун жалпыланган формасы**
then	else	
бирөө	бирөө	If туюнтма then оператор else оператор
бир нече	бирөө	If туюнтма then begin оператор; оператор; . . . оператор; end else оператор
бирөө	бир нече	If туюнтма then оператор else begin оператор; оператор; . . . оператор end
бир нече	бир нече	If туюнтма then begin оператор; оператор; . . . оператор; end else begin оператор; оператор; . . . оператор end

If операторун туура пайдалануудагы дагы бир синтаксистик татаалдык камтылуучу if операторун жазууда пайда болот. Эгер камтылуучу if оператору кандайдыр бир курама оператордун пределинде жайгашкан болсо, анда бул учурда анчалык проблема жок, анткени begin жана end кызматчы сөздөрү ар бир альтернативанын аракет этүү областын так чектеп коёт.

Мисал.

```
if  туюнтма
      then begin
             оператор;
             if  туюнтма
                then оператор
                else  оператор
             оператор
             end
     else  begin
             оператор;
             if  туюнтма
                 then оператор
             end
```

Бирок, эгерде камтылуучу if оператору альтернативанын бутагындагы жалгыз оператор болсо, анда else бутагы if операторун кайсы бирөөсүнө тийиштүү деген бир манилүү эместик пайда болушу мүмкүн.

Мисалы,

```
if  туюнтма  then
if  туюнтма  then
       оператор
       else    оператор
```

Мындай учурда төмөнкү эрежени эсте туту керек.

else кызматчы сөзү али else кызматчы сөзү менен байланыша элек жакын турган if кызматчы сөзү менен байланышат.

Эгерде жогорку мисалда камтылуучулук структурасын так белгилесек, else сөзүн ага тура келүүчү then менен бир деңгээлде жазсак, анда төмөнкүдөй фрагментти алабыз.

```
  if  туюнтма
       then
        if  туюнтма
          then  оператор
         else    оператор
```

Жогоруда баяндалган нерселер үчүн резюме катары төмөнкү мисалды келтиребиз.

```
Program if_then_else;
  var
     Priviledge, class_C, Want:  Byte;
     Begin
         Write (‘Сизде айдоочунун күбөлүгү барбы (0-жок/1-бар)?’);
          Readln (Priviledge);
          if Priviledge = 1
                then
                  Begin
                    Write (‘С-классыбы (0-жок/1-бар)?’);
                     Readln(class_C);
                  End;
                else
                  Begin
                     Write (‘Сиз машина алгыңыз келеби (0-жок/1-бар)?’);
                     Readln(Want);
                  end
        end.
```

CASE тандоо оператору

Биз карап өткөн if шарттуу оператору логикалык туюнтманын маанисинен көз каранды түрдө мүмкүн болгон эки аракеттин бирөөсүн гана тандоого мүмкүнчүлүк берет.

Case тандоо оператору if шарттуу операторунун жалпыланышы болуп ал кайра туташтыргычтын (переключатель) маанисинен көз каранды түрдө бир нече аракеттердин ичинен бирөөсүн аткарууга мүмкүнчүлүк берет.

Кайра туташтыргыч катары case жана of сөздөрүнүн ортосуна жайгашуучу туюнтма пайдапанылат. Бул туюнтманын жыйынтыгы элементтеринин жалпы саны 65535 тен ашпаган иреттик типтеги гана маани болушу мүмкүн.

Тандоо операторунун жалпы структурасын толук формада төмөнкүчө көрсөтүүгө болот.

```
Case   кайра_туташтыргыч   of
          константалар_тизмеси1:  Оператор1;
          константалар_тизмеси2:  Оператор2;
                        .    .    .
          константалар_тизмеси N:  ОператорN;
                                  else  ОператорE
end
```

Бир сөз менен тандоо операторунун иштөө логикасын мындайча баяндоого болот: Кайра_туташтыргычтын эсептелген мааниси Операторлордун ичинен кайсынысы аткарылышы керек экендигин аныктайт. Эгер Кайра_туташтыргыч Константалар_тизмеси1 маанилеринин ичинен бирөөсүн кабыл алса, анда Оператор1 аткарылат, ал эми калган бардык Операторлор каралбай өтүп кетет. Эгер Кайра_туташтыргыч Константалар_тизмеси2 маанилеринин ичинен бирөөсүн кабыл алса, анда Оператор2 аткарылат. Ушул сыяутуу Оператор3 төн ОператорN ге чейинки операторлор аткарылат.

Эгер Кайра_туташтыргыч тын мааниси Константалар_тизмеси1 ден N ге чейинки маанилердин ичинен бирөөсү менен да дал келбесе, анда бул учурда ОператорE аткарылат.

Case операторунун толук эмес формасында өзүнөн мурда келген тандоонун бардык варианттарына альтернативдүү болгон else бутагы жок болот. Бул учурда, эгер Кайра_туташтыргыч тын мааниси тандоо константаларынын маанилеринин бирөөсү менен да дал келбесе, анда case тин ичинде баяндалган операторлордун бири да аткарылбайт жана башкаруу case операторун туюктоочу end кызматчы сөзүнөн кийин турган операторго түздөн-түз берилет.

Case операторун пайдаланууунун мүнөздүү жолдорун демонстрациялоочу төмөнкү мисалдарды келтиребиз.

Мисал1.

```
Var   Arrow: Char;
      Position: record
            X, Y: Word
             end;
          . . .
      Begin
          . . .
      With Position do
        case Arrow of
                #72:    Y:= Y-1;
                #80:     Y:= Y+1;
                #75:      X:=X-1;
                #77:      X:=X+1
        end
          . . .
      end.
```

Мисал2.

```
Var   Symbol: Char;
   . . .
Begin
   . . .
Case Symbol  of
‘0’ .. ‘9’ :   writeln(‘бул цифра’);
‘a’ .. ‘z’ :   writeln(‘бул кичине тамга’);
‘A’ .. ‘Z’ :   writeln(‘бул чоң  тамга’);
#10, #13, #26:  writeln(‘бул башкаруучу символ’);
        Else writeln(‘бул башка символ’)
 end
                . . .
        end.
```

Мисал3.

```
Var Year: integer;
   . . .
   Begin
            . . .
   Case Year  of
        -500..-400:  writeln (‘ байыркы грек абагы’);
        480..500: Begin writeln (‘ Индияда болгон ондук’);
                        writeln (‘ эсептөө системасындагы’);
                        writeln (‘ биринчи жазуулар’);
                  end;
          1642: writeln (‘ Б. Паскалдын суммалоочу машинасы’);
                Else writeln (‘ жана башка, жана башка…’);
           end;
     . . .
   end.
```

5.2.3. Кайталоо операторлору

Turbo Pascal тилинин кайталоо операторлору кайталоонун төрт классикалык башкаруу конструкцияларынын ичинен үчөөсү менен түздөн-түз иш жүргүзөт. Алар: алдын ала шарттуу (с предусловием) цикл (while), шарты акырында берилүүчү (с постусловием) цикл (repeat) жана эсептегичтүү (со счетчиком) цикл (for) операторлору.

While алдын ала шарттуу цикл оператору

while оператору case оператору менен бирге Turbo Pascal тилиндеги эң универсалдуу башкаруу конструкциялары болуп эсептелинет. Эгер case операторунун жардамында каалагандай татаалдыктагы бутактануу шартын жазууга мүмкүн болсо, while операторунун жардамында каалагандай циклдик аракетти жазууга болот.

Алдын ала шарттуу цикл операторунун жалпы структурасын жөнөкөйлөтүп төмөндөгүдөй эки жолдун бири менен сүрөттөөгө болот.

Эгерде циклдин телосу бир оператордон турса	Эгерде циклдин телосу бирден көп операторду өзүнө камтыса

while Шарт do
Оператор
	while Шарт do
	begin
	оператор;
	оператор;
	.....
	оператор
	end

While оператору кандайдыр бир берилген Шарттан көз каранды түрдө бир эле аракеттердин тобун көп жолу аткарууга мүмкүнчүлүк берет. Ал Шарт – while жана do кызматчы сөздөрүнүн ортосуна жазылып, бульдук типтеги туюнтма болушу керек, б.а. True же False маанилерин гана кабыл алышы керек.

While циклинин ишин бир кыйла кеңири баяндап өтөлү. Адегенде циклдин ичине кирердин алдында Шарттын мааниси эсептелет. Эгер ал маани False болсо, циклдин ичине кирүү аткарылбайт да башкаруу циклдин телосундагы операторлордон кийин турган операторго берилет. Эгер Шарттын мааниси True болсо, анда циклдын ичине кирүү жана анын телосундагы Операторлорду бир ирет аткаруу болуп өтөт. Андан соң башкаруу кайрадан циклдин бөркүнө берилип, андагы Шарттын мааниси кайрадан эсептелинет. Эгерде Шарттын мааниси дагы эле True болсо (анын мааниси циклдин Операторлорунун алдынкы ирет аткарылыш убагында өзгөрүлмөлөрдүн маанилеринин өзгөрүшүнөн көз каранды), анда циклдын телосу дагы бир ирет аткарылыт жана дагы ушул сыяктуу улана берет. Качан гана Шарттын маанисин кезектеги эсептөөнүн жыйынтыгы False мааниси берери менен циклдин иши аяктайт.

Биз жогоруда санап өткөн while, repeat жана for цикл

операторлорунун касиеттерин салыштыруу жана алардын ар биринин ишин баяндоо максатында, бир өлчөмдүү n элементтен турган массивдин элементеринин суммасын эсептөөчү бир эле программанын фрагментин карайбыз.

While операторунун жардамында бул фрагментти төмөндөгүчө жазууга болот.

Мисал.

```
const n=20
begin
    ...
summa:=0; i:=1; {циклдин шарттарында турган өзгөрмөлөрдүн}
                { баштапкы маанилери берилген болушу керек}
    while i<=n do
        begin
            summa:=summa+A[i];
              i:=i+1; {циклдын телосундагы операторлордун жок   }
                      {дегенде бирөөсүндө  циклдын шартындагы   }
                      {өзгөрүлмөлөрдүн  маанилеринин өзгөрүшү   }
                      {болуп өтүп, ал барып-барып шарттын       }
                      { жалган болушуна  алып келиши керек      }
        end;
      ...
end.
```

Шарты аягында келген repeat цикл оператору

Шарты аягында келген цикл оператору repeat кызматчы сөзүнөн туруп анын артынан циклдин телосунун операторлору жайгашат, андан соң циклди туюктоочу until кызматчы сөзү келип бул сөздөн кийин циклдин аяктоо шарты көрсөтүлөт.

Repeat операторунун жалпы структурасын жөнөкөй түрдө төмөнкүчө көрсөтүүгө болот.

```
repeat
    Оператор;
    Оператор;
    - - - - - - - -
   Оператор;
    until  Шарт
```

Мындан көрүнүп тургандай repeat оператору while операторунан айырмаланып телодо бирден көп сандагы операторлорду жазуу зарыл болуп калганда курама операторду пайдалануну талап кылбайт.

Андан сырткары repeat оператору while операторунчалык универсалдуу эмес, анткени анын телосунун операторлору while циклинен айырмаланып дайыма жок дегенде бир жолу аткарылат. Бул болсо циклдин телосу бир да жолу аткарылбашы керек болгон учурларда repeat операторун циклди жазуу үчүн пайдаланууга болбойт дегендикти билдирет.

Циклдин аяктоо шарты катары пайдаланылуучу туюнтма бульдук типтеги жыйынтыкты бериши керек.

Repeat цикл операторунун ишинин жалпы принциби алдын ала шартуу цикл операторунун иштөө принцибине окшош, айырмасы - Шарт циклдин телосунун аткарылышынан мурун эмес, андан кийин текшерилет.

Repeat циклин башкаруу while циклин башкарууга тикеден тике карама-каршы б.а. цикл Шарттын мааниси False болуп турганда улантыла берет, качан Шарт True болору менен аяктайт.

Эми жогоруда келтирилген программанын фрагментин repeat операторунун жардамында жазалы.

```
const n=20;
    begin
      . . .
    summa:=0;
     i:=1; {Циклдин шартында турган өзгөрүлмөлөрдүн}
          {баштапкы маанилери берилген болушу керек}
       repeat
       summa:=summa+A[i];
       i:=i+1; {Циклдин телосундагы операторлордун жок}
              {дегенде бирөөсүндө циклдин шартынын }
              {өзгөрүлмөлөрүнүн өзгөрүшү болуп ал барып-}
              {барып шарттын чын болушуна алып келиши керек}
until i>n
  . . .
end.
```

Эсептегичтүү (параметрлүү) for цикл оператору

Программалоодо эсептегичтүү цикл операторун циклдин аткарылышына чейин циклдин кайталанышын эсептөөчү эсептегичтин баштапкы жана акыркы маанилари белгилүү болгон циклдик фрагменттер үчүн пайдалануу ылайыктуу. Бул while жана repeat универсалдык операторлоруна караганда анын колдонуш сферасынын анчалык кең эмес экендигин шарттайт. Бирок, качан аны

колдонуу мүмкүн болгон учурда for оператору өзүнүн эң сонун көрсөтмөлүүлүгүнөн улам шарттуу цикл операторунун жанында бир топ артыкчылыкка ээ болору шексиз.

Эсептегичтүү цикл операторунун жалпы структурасын жөнөкөй түрдө төмөнкү эки жолдун бирөөсү аркылуу көрсөтүүгө болот.

1) Эгер Эсептегич циклдин аткарылышында өзүнүн маанисин арттырса:

```
for Өзгөрүлмө := ЭсептегичтинБаштапкыМааниси
    to  ЭсептегичтинАкыркыМааниси
    do оператор
```

Мында оператор жок дегенде бир жолу акарылышы үчүн ЭсептегичтинБаштапкыМааниси ЭсептегичтинАкыркыМаанисинен **чоң эмес** болушу керек.

2) Эгер циклдин аткарылышында Эсептегич өзүнүн маанисин кемитсе:

```
for Өзгөрүлмө := ЭсептегичтинБаштапкыМааниси
                 downto ЭсептегичтинАкыркыМааниси
                 do оператор
```

Мында да оператор жок дегенде бир жолу аткарылышы үчүн ЭсептегичтинБаштапкыМааниси ЭсептегичтинАкыркы Маанисинен **кичине эмес** болушу керек.

Мисал.

```
begin
    . . .
  summa:=0;
      for i:=1 to n do
          summa:=summa+B[i];
    . . .
end.
```

for оператору үчүн, while жана repeat операторлорунан айрымаланып циклдин кайталаныш санын эсептөөчү ЭсептегичтинБаштапкыМаанисин (i:=1) берүү циклдин бөркүнүн алдында эмес бөрктүн ичинде аткарылат.

Андан сырткары циклдин телосу аяктагандан кийин эсептегичтин маанисинин өсүшү же кемиши автоматтык түрдө болуп өтөт. Ошон үчүн ЭсептегичтинМаанисин арттыруучу атайын оператор (i:=i+1) талап кылынбайт.

Катасы бар мисал:

```
begin
   . . .
    summa:=0:
       for i:=1 to n do
    begin
      summa:=summa+A[i];
    i:=i+1; ◄——— Ката!
    end;
   . . .
end.
```

Turbo Pascal тилинде анын башка бардык реализацияларындагыдай for оператору эки олуттуу чектөөлөргө ээ:

1) Эсептегичтин өзгөрүү кадамы же +1 (эгер to кызматчы сөзу пайдаланылса) же -1 (эгер downto кызматчы сөзу пайдаланылса) гана болушу мүмкүн.

2) Циклдер эсептегичи катары кирүүчү өзгөрүлмө иреттик типтеги гана болушу мүмкүн жана for оператору жайгашкан блок үчүн локалдык болушу керек.

While, repeat жана for операторлорунун иштерин салыштыруу

Каралган циклдик операторлор менен иштөөдөгү алардын айырмачылыктарын жана өзгөчүлүктөрүн белгилеп өтөбүз.

Алдын ала шарттуу цикл while (азырынча шарт чын)	**Шарты аягында келген цикл repeat (шарттын чын болушуна чейин)**
1) Циклге туура кирүү үчүн циклдин башталышына чейин циклдин шартын башкаруучу өзгөрүлмөлөрдүн баштапкы маанилери коюлган болушу керек.	
2) Итерациялардын белгилүү бир санынан кийин цикл аяктасын үчүн циклдин телосунда шартты башкаруучу өзгөрүлмөлөрдүн маанилерин өзгөртүүчү оператор катышып турган болушу керек	
3) Азырынча шарт чын (азырынча true) болуп турганда цикл иштейт	3) Азырынча шарт жалган (азырынча false) болуп турганда цикл иштейт
4) Качан шарт жалган болуп калганда (false ке чейин) цикл аяктайт	4) Качан шарт чын болуп калганда (true ге чейин) цикл аяктайт
5) Эгер циклге кирерде шарттын баштапкы мааниси false ге барабар болсо, цикл бир ирет да аткарылбастыгы мүмкүн	5) Цикл жок дегенде бир ирет сөзсүз аткарылат
6) Эгер циклдин телосунда бирден көп сандагы оператор талап кылынса, анда курама операторду пайдалануу зарыл	6) Циклдин телосундагы операторлордун санынан көз карандысыз курама оператордон пайдалануу талап кылынбайт

Эсептегичтүү for цикли
1) Циклдин бөркүнө чейин эсептегич - өзгөрүлмөнүн баштапкы маанисин берүү талап кылынбайт
2) Циклдин бөркүндө турган өзгөрүлмөлөрдүн маанилерин циклдин телосунда өзгөртүүгө уруксат берилбейт
3) Циклдин итерацияларынын саны өзгөрбөс жана циклдин төмөнкү жана жогорку чек аралары жана кадамы аркылуу так аныкталат
4) Циклдин ишинин нормалдуу жүрүшү goto же break жана continue операторлору аркылуу бузулушу мүмкүн
5) Эгер циклдин кадамы төмөнкү чек арадан жогорку чекти көздөй эмес, карама-каршы багытта өзгөрсө, анда цикл бир да жолу аткарылбастыгы мүмкүн

While, repeat жана for циклдерин жазууларды салыштыруу көрсөтмөлүү болушу үчүн алар үчүн жогоруда караган фрагменттерди бир жерге топтойлу.

```
s:=0;
 i:=1;
while i<=n do
begin
    s:=s+a[i];
    i:=i+1;
end.
```

```
s:=0;
:=1;
repeat
    s:=s+a[i];
    i:=i+1;
until  i>n;
```

```
    s:=0;
for i:=1 to n do
    s:=s+a[i];
```

With бириктирүү (присоединение) оператору

With бириктирүү оператору жазуулардын (record) талааларына кайрылууну жөнөкөйлөтүү үчүн арналган.

Эгер With операторун пайдаланбасак, анда жазуулардын талааларына кайрылууда, чекиттер менен ажыратылган идентификаторлордун чынжырчаларынан турган талаанын толук квалификацияланууну атын көрсөтүү зарыл.

Мисалы, эгерде төмөнкүдөй баяндоо жасалса,

```
type T_Rec =record
                A: record
                     i:=record
                          x:= char:
                          y:=byte
                        end
                   end
                c: real
              end
 D: string
              end
Var Rec : T_Rec;
```

анда У талаасына О маанисин, ал эми С талаасын 3,1415 маанисин ыйгаруу үчүн

```
Rec.A.B.Y   :=O;
Rec.A.C     :=3.1415;
```

деп жазуу зарыл.

Жөнөкөй учурда бириктирүү оператору жазуунун талааларынын аттарын төмөндөгүдөй кылып кыскартууга мүмкүнчүлүк берет.

```
with Rec do
begin
    A.B.Y   :=0;
    A.C     :=3.1415
end
```

Эгерде бир эмес эки with операторун пайдаланылса, анда мындайча жазуу мүмкүн

```
with Rec do
  with A do
  begin
      B.Y   :=0;
      C     :=3.1415
   end
```

Акыркы фрагментти Rec жана A аттарын шилтемелердин(ссылки) бир тизмесинде көрсөтүп with оператору аркылуу бир кыйла компакттуу түрдө жазууга болот.

```
with Rec, A do
begin
     B.Y   :=0;
     C     :=3.1415
End
```

Эгер X жана У талааларына гана кирүү талап кылынса, анда төмөндөгүчө жазсак болот

```
with Rec, A, B do
begin
       X  := '*' ;
       Y :=0
end
```

6. МОДУЛДАР

Аз бил, бирок пайдалан,
Аз айт, бирок аткар.
(И. Гете)

Модулдар ар кандай программаларды талдап чыгууда принциптердин ичинен негизгиси – информацияны жашыруу (информацияны бекитүү - information hiding) болгон модулдук программалоонун принциптерин ишке ашыруу үчүн арналган. Бул принципке ылайык программанын логикалык көз каранды эмес фрагменттеринин өз-ара таасир этүүлөрү минимумга келтирилиши керек. Модулдарда реализацияланууучу информацияны жашыруу принциби ишенимдүү иштөөчү жана жеңил модификацияланууучу программаларды түзүүгө мүмкүнчүлүк берет. Turbo Paskal тилинде модулдар көбүнчө процедуралардын/функциялардын жана объекттердин библиотекаларын түзүү үчүн колдонулуп кийин пайдаланууучу аларды өзү иштеп чыккан программаларда пайдалана алат.

Unit жана модул терминдери синоним терминдер болуп ар түрдүү тилдерде бир эле түшүнүктү билдирет. Unit концепциясы Н.Вирт тарабынан иштелинип чыгылып, жогорку деңгээлдеги Модула-2 тили үчүн өнүктүрүлгөн. Бирок, бул термин жана программаларды иштеп чыгуунун бул жолу башка тилдерде да реализацияланган.

Модулдарды пайдалануда алардын аттарын тура көрсөтүү негизги мааниге ээ.

Стандарттык модулдарды кошууда (пайдаланганда) uses сүйлөмүндө алардын идентификаторлорун туура (корректтүү) жазуу жетиштүү.

Программист өзүнүн менчик модулдарын иштеп чыгууда төмөнкүдөй өзгөчөлүктөрдү эсте тутуусу зарыл:

- бирдей атка ээ болгон модулдарды бир мезгилде пайдаланууга болбойт;
- бөрктө (unit) көрсөтүлгөн модулдун идентификатору баштапкы (.PAS) жана объекттик (.TPU, .TPP, .TPW) кодду кармап турган файлдардын аттары менен дал келиши керек;
- эгерде модулдун идентификатору сегиз символдон узун болсо, анда ал файлдардын аттары менен биринчи сегиз символу боюнча дал келиши керек.

Төмөнкү таблицада модулдун ар бир бөлүгүнүн мааниси жана кызматын баяндаган комментарийлер менен толукталган модулдун жалпы структурасын келтиребиз.

```
unit  Модулдун_идентификатору;

                { Интерфейстик бөлүк }

    interface

      {Бул бөлүктө берилген модулдун башка пайдалануучулук жана}
      {стандарттык модулдар менен өз ара аракети, ошондой эле   }
      {башка программа менен өз ара аракети баяндалат. Башкача  }
      { айтканда модулдун «тышкы дүйнө» менен болгон             }
      {өз ара аракети баяндалат.                                 }

            {Интерфейстик бөлүктүн импорт тизмеси}

    uses

       {Бул тизмеде үтүр аркылуу интерфейстик бөлүктөрүндөгү}
       { информацияларга  ушул модулдун ичинде кайрылууга    }
       { мүмкүн болгон модулдардын идентификаторлору          }
       { саналып көрсөтүлөт.                                  }
       { Бул жерде алынган информация берилген модулдун       }
       { interface  бөлүгүнүн баяндоолорунда пайдаланыла      }
       { турган  модулдардын  гана  идентификаторлорун        }
       { баяндоо максатка ылайыктуу.                          }

            {Интерфейстик бөлүктүн экспорт тизмеси}

const      {Экспорт тизмеси ушул модулда аныкталып  бирок          }
type       { өзүнүн  uses жолчосунда ушул модулдун атын кармап     }
var        { турган башка бардык модулдарда жана программаларда    }
procedure  { пайдалануга уруксат берилген константаларды,           }
function     { өзгөрүлмөлөрдүн типтерин баяндоо бөлүмчөлөрүнөн,     }
             { процедуралардын жана функциялардын бөрктөрүнөн       }
           { турат. Процедуралар жана функциялар үчүн   бул        }
             { жерде бөрктөр гана баяндалат, бирок сөзсүз формалдык }
             { параметрлердин толук баяндалышы менен.                }

                {Реализация бөлүгү}

implementation  {Бул бөлүктө берилген модулдун баяндоолорунун  }
                { башка модулдар жана программалар үчүн         }
                  { уруксат берилген реализациялык  (өздүк) бөлүгү }
                  { көрсөтүлөт. Башка сөз менен айтканда модулдун }
                  {«ички ашканасы»  көрсөтүлөт.                   }
```

```
{Реализация бөлүгүнүн  импорт тизмеси}
uses
      {Бул тизмеде үтүр аркылуу интерфейстик бөлүктөрүндөгү    }
      { информацияларга  ушул модулдун ичинде кайрылууга       }
      { мүмкүн болгон модулдардын идентификаторлору саналып  }
      { көрсөтүлөт.                                                    }
      { Бул жерде алынган информация ущул  модулдун interface }
      {бөлүгүнүн баяндоолорунда пайдаланылбаган жана алардын }
      {пайдаланылгандыгы жөнүндө бир да башка модул билбеши }
      {керек болгон  бардык   зарыл болгон модулдардын          }
      { идентификаторлорун  баяндоо максатка ылайыктуу.        }

{Модулдар үчүн ички баяндоолордун бөлүмчөлөрү}
label      {Бул бөлүмчөлөрдө берилген модул тарабынан аткарылуучу}
const      { алгоритмдик аракеттерди баяндоочу жана   берилген      }
type       { модул үчүн  гана «өздүк менчик» болуп эсептелген эн     }
var        { -белгилер (меткалар), константалар, типтер, өзгөрүлмөлөр,}
procedure  { процедуралар жана функциялар баяндалат.                 }
function    { Бул баяндоолорго бир да башка модул  үчүн уруксат жок.}
           { Бул бөлүмчөлөрдө процедуралардын жана функциялардын,}
            { бөрктөрүн формалдык параметрлердин тизмеси жок        }
            { көрсөтүүгө уруксат берилет. Эгерде бары бир бөрктөр   }
            { параметрлер менен көрсөтүлүп калса, анда формалдык    }
            { параметрлердин тизмеси interface бөлүгүндөгү тиешелүү  }
            { процедуранын (функциянын) ушундай тизмеси менен       }
            { дал келиши керек.                                      }

{ Инициалдаштыруу бөлүгү}
begin  {Бул бөлүмдө модулдун туура (корректно) иштешин ишке      }
       { салу үчүн зарыл болгон баштапкы коюулардын (установка)  }
       { операторлору көрсөтүлөт.                                 }
       { Программада пайдаланылуучу модулдардын инизиалдаш-      }
       {тыруу, бөлүмүнүн операторлору программанын алгачкы ишке }
       {салынышында  uses  сүйлөмүндө модулдардын идентифи-      }
        { каторлору кандай тартипте баяндалган болсо ошондой      }
        { тартипте аткарылат.}
        { Эгерде инизиалдаштыруу операторлору талап кылынбаса,    }
        { анда Begin кызматчы сөзү алынып коюлган болушу мүмкүн. }
end.
```

Пайдалануучу өзүнүн менчик сервистик процедуралар жана функциялар библиотекасын түзүүгө карата модулдарды пайдалануунун жөнөкөй демонстрациялык мисалын келтиребиз. Бул мисалда модулдар менен негизги (башкы) программанын ортосундагы өз ара байланыштардын мүнөздүү структурасы көрсөтүлгөн. Модулдар менен иштөөдө алардын процедуралардан жана функциялардан болгон негизги айырмачылыктарын эсте тутуу зарыл:

Модулдар үчүн глобалдык жана локалдык өзгөрүлмөлөрдүн аракет этүү сферасынын традициялык эрежелери иштебейт. «Модул» тилдик конструкциясы негизги программада жарыяланган глобалдык өзгөрүлмөлөрдүн модулдун ички баяндоолоруна таасир этүүсүн жоготуу үчүн атайын иштелип чыгылган.

Ошондуктан эгер бары бир программанын бардык блоктору үчүн тиешеси болгон кандайдыр бир баяндоолорду киргизүү зарыл болуп калса (б.а. глобалдык баяндоолорду), анда модулдар үчүн бул нерсени бир гана жол менен жасоого болот. Ал жол – глобалдык жарыялоолордун атайын модулун түзүү (биздин мисалда ал - Globals) жана аны калган бардык модулдардын жана башкы (негизги) программанын импорттор тизмесине киргизүү.

```
unit  Globals;
            {Глобалдык баяндоолордун модулу}
interface
     const   len = 100;
     type
          T_Vector = array [ 1 .. len ] of  integer;
implementation
end.
```

```
unit VectorService;
   {Бул -узундугу  100 элементке чейин болгон бир өлчөмдүү  }
   {массивдер менен иштөө үчүн сервистик процедураларды }
   {жана функцияларды камтыган библиотекалык модул        }
interface
            {Импорттолуучу модулдардын тизмеси}
uses Globals;
          { Globals модулун гана көрсөтүү жетиштүү,          }
          { анткени  interface  бөлүгүнүн жарыялоолорунда  }
          {  башка стандарттык жана пайдаланучулук          }
          {модулдардан алынган информациялар                }
```

```
                { пайдаланылбайт.                                    }
                { interface  бөлүгүндө баяндалган модулдарга         }
                { implementation  де да кайрылууга болот.            }

                {Экспорттолуучу процедуралардын тизмеси}
    procedure Vect_Max (Vect: T_Vector; N:Byte);
                  {Массивдин максималдык элементин табуу}
                   {жана аны  печатка чыгаруу процедурасы}

    procedure Vect_Min( Vect: T_Vector; N:Byte);
                  { Массивдин минималдык элементин табуу }
                   { жана аны  печатка чыгаруу процедурасы }

    procedure Vect_Inverse (Vect: T_Vector; N:Byte);
                 {Масиивдин элементтерин тескери тартипте }
                          { төңкөрүү  процедурасы }

implementation
                    {Модулдун «өздүк» импорт тизмеси}
   uses Crt; { Каралып жаткан мисал үчүн бул жерде Crt  модулун }
                {гана көрсөтүү зарыл. Globals модулунан алынган      }
                { информация implementation де пайдаланылганы    }
               {менен аны бул тизмеде көрсөтүүгө болбойт, анткени }
               {ар бир модул же interface те же implementation  де  }
               {бир гана жолу көрсөтүлүшү керек.                    }
               { implementation  дин ордуна CRT модулун             }
               { interface те баяндоо мүмкүн эле.  Бирок бул         }
               { модулдарды жазуудагы жаман стил деп эсептнлинет,}
               { анткени Crt ден алынган информация                 }
               { interface теги баяндоолордо пайдаланылбайт.        }

      pocedure Print_Vect (Vect: T_Vector ; N:Byte);
  { Модул үчүн вектордун элементтерин печатка чыгаруучу  }
  { ички процедура.  Ички процедуралардын жана             }
  { функциялардын бөрктөрү  ар дайым толук                 }
  {  (параметрлери менен ) баяндалат.                     }

      var i: byte;
        begin
            writeln (‘Вектордун элементтери:’);
            for i := 1 to N do write (Vect[i]:5,’  ‘);
           writeln;
        end;
             {!!! interface бөлүмүндө процедуралардын жана   }
             {!!! функциялардын бөрктөрү эбак баяндалган,    }
             {!!! реализация бөлүгүндө аларды параметрсиз    }
                        {!!! эле көрсөтө берсе болот. }
```

```
  procedure Vect_Max;
  var
    Max :  Integer;
     i       :  Byte;
  begin
       Max := Vect[1];
       for  i : = 2  to N  do
            if   Vect[i] > Max   then  Max : = Vect[i];
  ClrScr;
   Print_Vect ( Vect, N);
   writeln ('Вектордун максималдык элементи:' , Max)
  end;
 procedure Vect_Min;
 var
   Min : integer ;
    i      : byte ;
 begin
     Min := Vect[1];
     for  i :=  2  to  N  do
          if   Vect[i]  < Min  then  Min : = Vect[i];
  ClrScr;
  Print_Vect (Vect, N );
writeln ('Вектордун минималдык элементи:', Min)
end;

procedure Vect_Inverse ;
 var
     Temp :  Integer ;
      i         :  Byte ;
 begin
   ClrScr;
   writeln ('Төңкөрүлгөнгө чейинки вектор:');
    print_Vect (Vect, N) ;
       for i :=1  to  N  div  2  do
       begin    Temp := Vect[i] ;
                Vect[i] := Vect [N - i+1] ;
                 Vect[n-i+1] := Temp;
       end;
    writeln('Төңкөрүлгөндүн кийинки вектор:');
     print_Vect (Vect , N ) ;
 end;
            {Инициализация бөлүгү жок}
end.
```

```
program Main;
          {Негизги программа}
  uses  Crt, Globals, VectorService;
  var
     A    :  T_Vector ;
     i, N :  Byte;
  begin
    ClrScr;
    writeln (‘ Массивдин узундугун <=100  киргизгиле:’);
     readln (N);
    writeln (‘ Массивдин элементтерин киргизгиле:’);
    for i := 1 to N do  read (A[i]);  readln;
        Vect_Max (A,N);
         readln;  {Жыйынтыкты экранда Enter клавишасы    }
                  { басылганга чейин кармап турат.          }
       Vect_Min (A,N);
         readln;  {Жыйынтыкты экранда Enter клавишасы    }
                  { басылганга чейин кармап турат.          }
       Vect_Inverse (A,N);
   end.
```

Контролдук тапшырмалар

1) Узундугу 20 элементке чейин болгон бир өлчөмдүү массивдердин элементтерин өсүү, кемүү тартибинде жайгаштыруучу жана массивдердин берилген сандан кичине болгон элементтерин печатка чыгарууну ишке ашыруучу сервистик процедураларды жана функцияларды камтыган библиотекалык модулду түзгүлө.
2) Берилген интервалдагы натуралдык сандардын ичинен 3 кө эселүү, 7 ге эселүү сандарды табуучу жана аларды печатка чыгарууну ишке ашыруучу сервистик процедураларды жана функцияларды камтыган библиотекалык модулду түзгүлө.

7. ДИНАМИКАЛЫК БАЙЛАНЫШУУЧУ БИБЛИОТЕКАЛАР

Жакшылыкты бир гана акыл берет,
Билимден урмат, атак кошо келет.
(Ж. Баласагын)

Динамикалык байланышуучу (кошулуучу (связываемые)) библиотекалар – ***Dynamically Linked Libraries*** (DLL) - колдонмо (прикладный) программаларга DOS тун корголгон (зашищенный) режиминде жана Windows режиминде иштеген учурда кошумча мүмкүнчүлүктөрдү берет жана компилятордун BP.EXE, BPC.EXE, BPW.EXE версиялары тарабынан гана түзүлө алат. Программалык камсыздандырууну өндүрүп чыгаруучу фирмалар тарабынан иштелип чыгылган стандарттык DLLдер менен бирге, ***Borland Pascal with Objects*** программисттерге стандарттык DLL дер менен катар пайдалануга мүмкүн болгон менчик DLL дерди түзүүгө мүмкүнчүлүк берет.

Алгач динамикалык байланылуучу библиотекалар Windows каптамасында (оболочкасында) пайдаланылган. Borland Pascal with Objects DOS тун корголгон режими үчүн да DLL дерди иштеп чыгууга мүмкүнчүлүк берди. DLL дердин баалуулугу алар оперативдик эске жүктүлгөндөн кийин да бир нече башка колдонмо программалар менен биргеликте пайдаланыла алгандыгында болуп эсептелет.

DOS үчүн DLL дер DOS тун корголгон режиминде иштегени менен алар Windows үчүн DLL дер менен толук биргелешкен (совместимы). Бул касиет программаларды иштеп чыгуучуларга DOS то да, ошондой эле Wihdows да да иштей турган программаларды түзүүгө мүмкүнчүлүк берет.

Андан сырткары, DLL дер “көп тилдүүү” долбоорлор менен иш жүргүзөт:

Borland Pascal with Objects тилинде түзүлгөн программаларда башка тилдерде түзүлгөн DLL дерди пайдалануга уруксат берилет, ал эми башка тилде түзүлгөн программаларда болсо Borland Pascal with Objects те түзүлгөн DLL дерди пайдалануга уруксат берилет.

“Динамикалык байланышуучу библиотека” - тилдик конструкциясы бир жагынан “модуль” конструкциясынын улантылышы жана өнүктүрүлүшү болсо, экинчи жагынан ага салыштырмалуу кээ бир чектөөлөргө ээ.

DLL дердин өзгөчөлүктөрүн төмөнкү айырмачылыктарменен түшүндүрүүгө болот:

- Башкы программа менен модулдардын компоновкасы статикалык түрдө компиляция кезинде аткарылат, ал эми DLL дердики болсо динамикалык түрдө программанын аткарылыш убагында болот. Бул болсо бир убакта параллель иштеп жаткан бир нече программалар үчүн оперативдик эсте алардын ар биринде пайдаланылуучу процедуралардын жана функциялардын бирден гана нускасын кармап турууга мүмкүнчүлүк берет;
- DLL дин коду жана ресурстары, модулдан айырмаланып, аны пайдалануучу программа менен компоновкаланбайт, программа аткарылып жаткан кезде кайрылууга уруксат бериллүүчү **.**DLL кеңейтирилишине ээ болгон өзүнчө аткарылуучу файлда туруп турат. DLL дердин программа аркылуу чакырылуучу процедуралары жана функциялары ал программа менен динамикалык түрдө байланышышат.
- DLL дердин модулдарга салыштырмалуу чектелгендиги:
- Модулдар ар түрдүү тилдик бирдиктерди (типтерди, константаларды, өзгөрүлмөлөрдү ж.б.) экспорттой алат, ал эми DLL дер болсо процедураларды жана функцияларды гана. DLL өзгөрүлмөлөрдү кармап тура алганы менен модулдар аларды пайдалануу үчүн импорттой албайт. DLL дердин өзгөрүлмөлөрүнө болгон ар кандай кайрылуу процедуралык интерфейс аркылуу ишке ашырылышы керек.
- Пайдалануучу түзгөн программаны компиляциялоодо андагы пайдаланылуучу DLL дер модулдарга окшоп автоматтык түрдө компиляцияланбайт. DLL дерди өзүнчө компиляциялоо талап кылынат.

7.1. Динамикалык байланышуучу библиотекаларды түзүү

Динамикалык байланышуучу библиотекалардын структурасы иш жүзүндө так эле программанын структурасындай. Айырмасы – program кызматчы сөзүнүн ордуна library сөзү жазылат. Бул сөз компиляторго кеңейтирилиши .EXE эмес .DLL болгон DLL дин аткарылучу файлын түзүү талап кылынарын көрсөтөт.

Мисал катары «Модулдар» темасында каралган мисалдагы модул тарабынан аткарылуучу аракеттерди DLL көрүнүшүндө жазабыз.

```
library    Vector;
   uses  Globals, Crt;
   procedure  PrintVect (Vect : T_Vector ;  N: Byte);
   var   i : Byte;
     begin
       Writeln (' Вектордун элементтери: ');
       for i:= 1 to N do write (Vect [i] : 5, '   ' );
       WriteLn;
     end;

   procedure Vect _Max(Vect: T _Vector ; N :Byte) ; export;
    var  Max : Integer;
           i    :  Byte;
     begin
       Max:= Vect[1];
       for  i :=  2  to N do
       If Vect[i] > Max then
       Max:= Vect[i];
       Clrscr;
       PrintVect(Vect,N);
       Writeln (' Вектордун максималдык элементи:', Max);
    End;

procedure Vect _Min(Vect: T _Vector ; N :Byte); export;
    var Min: Integer ; i:  Byte;
   begin
     Min:= Vect[1];
    for  i :=  2  to  N  do
         If Vect[i] < Min then
         Min:= Vect[i];
         ClrScr;
    PrintVect (Vect, N);
    Writeln ('Вектордун минималдык элементи:', Min)
    end;

  procedure Vect _Inverse (var Vect: T_Vector; N: Byte); export;
    var  Temp: Integer;
      i     : Byte;
    begin
       ClrScr;
       writeln('Төңкөрулгөнгө чейинки вектор:');
       PrintVect  (Vect, N);
      for i:= 1 to  N div 2  do
```

```
        begin Temp:= Vect [i];
          Vect[i]:= Vect[N-i+1];
          Vect[N-i+1]:= Temp;
        end;
  writeln('Төңкөрүлгөндөн кийинки вектор:');
  PrintVect ( Vect, N);
end;
   exports
        Vect_Max   index  1,
        Vect_Min   index  2,
        Vect_Inverse    index  3;
   begin
    {  Инициализациялоо бөлүгү жок }
   end.
```

Динамикалык байланышуучу библиотекаларды түзүүдө негизги роль exports сүйлөмүнө (аягында «s» тамгасы бар) жана түзүлгөн DLL ден экспорттолуучу процедуралардын жана функциялардын бөрктөрүндө көрсөтүлүүчү export (аягында «s» тамгасы жок) процедуралык директивасына тандык. Export директивасы процедура же функция үчүн мажбур түрдө чакыруунун алыскы тибин пайдаланат жана аны процедура/функция үчүн кийрүү жана чыгаруунун атайын кодун генерациялоо менен экспорттоого даярдайт. Бирок, процедуранын/функциянын иш жүзүндөгү экспорту качан ал баяндалуучу библиотеканын exports сүйлөмүндө көрсөтүлмөйүнчө жүргүзүлбөйт.

Инициализация бөлүмүнүн операторлору модулдагыдай бир ирет аткарылат, бирок, программаны ишке салган учурда эмес, а библиотеканын алгачкы жүктөлүш учурунда аткарылышат. Оперативдик эсте DLL ди табуу талабын аныктоо үчүн ***DLL ди пайдалануу эсептегичи*** деп аталган эсептегич кызмат кылат. Ар бир DLL үчүн өзүнүн DLL ди пайдалануу эсептегичи түзүлүп ал убакыттын берилген моментинде аны канча программа пайдаланып жаткандыгын көрсөтөт. Динамикалык байланышуучу библиотека жүктөлгөндөн кийин анын пайдалануу эсептегичи нөлдөн чоң болуп турганда ал эсте сакталып тура берет. Качан ушул DLL ди пайдаланып жаткан бардык колдонмо программалар ишин аяктагандан кийин анын пайдалануу эсептегичи нөл болуп калат да DLL эстен жоголот. Эгер жүктөлүнүп калган кандайдыр бир DLL ди дагы бир программа пайдалана баштаса, анда анын пайдалануу эсептегичи чоңоет, ал эми инициализация бөлүмүнүн оператору кайталанып аткарылбайт.

7.2. DLL ден процедураларды жана функцияларды импорттоо

Borland Pascal with Objects тилинде түзүлгөн программалар жана модулдар DLL дерден прецедураларды жана функцияларды үч жол менен импорттай алат:

1. аты боюнча;
2. жаңы аты боюнча;
3. иреттик номери боюнча.

Процедураны же функцияны импорттоо факты анын бөркүн, программанын же модулдун тиешелүү жарыялоолор бөлүктөрүндө баяндоо аркылуу көрсөтүлөт. Бул учурда DLL ден импорттолуучу процедуралардын жана функциялардын бөрктөрү төмөндөгү талаптарга жооп бериши керек:

- процедуралык external директивасы пайдаланылышы керек;
- DLL дин аты external директивасынан кийин көрсөтүлөт;
- far процедуралык директивасы же компилятордун {$F+} директивасы менен берилген чакыруунун алыскы модели (дальняя модель) пайдаланылышы керек.

7.2.1. Аты боюнча импорттоо

Процедураны же функцияны анын өздүк аты (индентификатору) боюнча импортоо анын бөркүн баяндоодо index жана name директивасы жок болгон учурда аткарылат.

Мисалы,

```
Procedure MyProc; external 'MYLIB';
```

Бул баяндоодо MyProc процедурасы MYLIB динамикалык байланышуучу библиотекасынан өзүнүн өздүк MYPROC аты боюнча импорттолот.

7.2.2. Жаңы аты боюнча импорттоо

Эгер процедуранын же функциянын бөркүндө **name** директивасы көрсөтүлгөн болсо, анда ал өзүнүн аты боюнча эмес, **name** де көрсөтүлгөн ат боюнча импорттолот.

Мисалы,

```
procedure MyProc; external 'MYLIB' ; name 'NEWNAME'
```

Бул баяндоодо MyProc процедурасы MYLIB динамикалык байланышуучу библиотекасынан өзүнүн өздүк MYPROC аты боюнча эмес, а жаңы NEWNAME аты боюнча импорттолот. Жаңы ат катары каалагандай жолчолук туюнтма -константаны пайдаланууга болот.

7.2.3. Иреттик номери боюнча импорттоо

Эгер процуедуранын же функциянын бөркүндө index директивасы көрсөтүлгөн болсо, анда ал өзүнүн аты боюнча эмес index те көрсөтүлгөн иреттик маани боюнча импорттолот. Иреттик маани катары ар кандай бүтүн туюнтма -константаны пайдаланууга болот. Мындайча импорттоо модулдун жүктөлүш убактысын азайтууга мүмкүндүк берет, анткени DLL деги аттардын таблицасынан керектүү атты издөө зарылдыгы жоголот.

Мисалы,

```
procedure MyProc; external 'MYLIB' index 7;
```

Бул баяндоодо MYLIB динамикалык байланышуучу библиотека-сынан MyProc процедурасы 7 деген индекс боюнча импорттолот.

7.2.4. Импорт модулдары

Импорт модулдары программаны структуралоону жана анын иштелип чыккандан кийинки коштолушун жакшыртуу максатында түзүлөт. Мындай модуль DLL ден импорттолуучу процедуралардын жана функциялардын, ошондой эле DLL менен болгон интерфейс үчүн зарыл болгон бардык типтердин жана константалардын баяндамаларынын модулдун структурасынын ичине алынган топтому болуп эсептелет. Импорт модулдарынын болушу DLL менен болгон интерфейс үчүн сөзсүз шарт эмес, бирок алар DLL дер пайдаланылуучу программаларды иштеп чыгууну бир кыйла жөнөкөйлөтөт.

Мисал катары жогорто баяндалган VECTOR.DLL файлында турган динамикалык байланышуучу библиотекадан импорттолуучу процедуралардын MyVector модулуна бириктирилишин карайбыз.

```
unit MyVector;
  interface
    uses Globals;
    procedure  Vect_Max(Vect :  T_Vectorc ; N:  Byte);
    procedure Vect_Min(Vect:   T_Vector ; N:  Byte);
    procedure  Vect_Inverse (var Vect :  T_Vector ;  N:  Byte);
```

```
imple mentation
   procedure Vect _ Max ;  external   'VECTOR'    index 1;
   procedure Vect_ Min;  external   'VECTOR'   index 2;
   procedure Vect_Inverse ;  external   'VECTOR'   index 3;
end.
```

Бул модулдун жана VECTOR.DLL библиотекасынын иштөөсүн текшерүү үчүн, мисалы, төмөнкү программа пайдаланылышы мүмкүн:

```
program Main;
      uses Crt , Globals, MyVector;
     var  A: T_Vector;
          i , N :  Byte;
  begin
     ClrScr;
    writeln ('Массивдин <=100 болгон узундугун киргизгиле:');
    readln (N);
    writeln (' Массивдин элементтерин киргизгиле:');
    for i:=1 to N do read (A[i]);
    readln ;
    Vect_ Max (A,  N);
    readln ;
    Vect_ Min (A,  N);
    readln ;
    Vect_Inverse (A,  N);
   end.
```

7.3. DLL деги өзгөрүлмөлөрдүн аракет этүү аймагы

Биз жогоруда белгилеп өткөндөй DLL модулдарга салыштырмалуу аны чакыруучу программалар жана модулдар менен өз ара аракет этүүдө бир кыйла чектелген мүмкүнчүлүктөргө ээ. DLL берилгендердин менчик (собственный) сегментинде жайгашып андагы баяндалган ар кандай өзгөрүлмө ушул DLL үчүн локалдык болот. Андан сырткары, DLL аны чакыруучу модулдардын ичинде баяндалган өзгөрүлмөлөргө кайрылууга уруксат албай жана башка модулдарга өзгөрүлмөлөрдү экспорттобой кое албайт.

***DLL** дин аны чакыруучу модулдар менен болгон бардык өз ара аракет этүүлөрү процедуралык интерфейс аркылуу гана реализацияланган болушу керек.*

7.4. DOS тун корголгон режиминде жана Windows да биргелешип пайдаланылуучу DLL дер

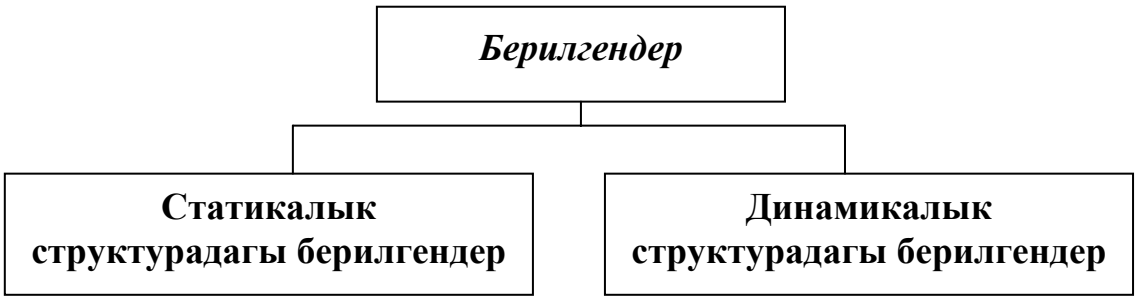
Бирге пайдаланылуучу DLL дер экилик коддун деңгээлинде биргелешкен болушат. Мындай DLL ди компиляциялоодо максаттык платформа (целевая платформа) катары Windows ду орнотуу талап кылынат. Мындай орнотуу интегралдашкан каптамада (интегрированная оболочка) Compile менюсундагы Target командасынын жардамында аткарылат. DOS тун корголгон режими үчүн компиляцияланган DLL, Windows каптамасынын башкаруусу алдында аткарыла албайт, анткени DOS тун корголгон режимин пайдалануучу системанын библиотекасы Windows да колдонбош керек болгон DOS жана DPMI деген өзүнчө функционалдык чакырууларды пайдаланат. Андан сырткары, биргелешип пайдаланылуучу DLL операциялык система менен, өзүнүн ичине DOSтун корголгон режими жана Windows үчүн жалпы болгон функцияларды камтыган WinAPI модулу аркылуу гана аракет эте алат.

8. БЕРИЛГЕНДЕР СТРУКТУРАСЫН КЛАССИФИКАЦИЯЛОО

Эмгектене билүү – адам үчүн түгөнгүс байлык.
(Эзоп)

Берилгендер структурасы алгоритмдер менен катар түзүлүүчү программалардын негизги курамдык (составдык) бөлүктөрү болуп саналат. Ошондуктан профессор Н.Вирт өзүнүн китептеринин бирин бекеринен "Алгоритмдер+Берилгендердин структурасы = Программа" деп атаган эмес.

Программалоодо пайдаланылуучу берилгендерди эки чоң тайпага бөлүүгө болот:



Статикалык структурадагы берилгендер – булар элементтеринин өз ара жайланышы жана өз ара байланыштары дайыма туруктуу бойдон кала берген берилгендер.

Динамикалык структурадагы берилгендер – булар ички түзүлүшү кандайдыр бир мыйзам (закон) боюнча калыптанган, бирок элементтеринин саны, алардын өз ара жайланышы жана өз ара байланыштары программа аткарылып жаткан мезгилде калыптануу мыйзамына ылайык динамикалык түрдө өзгөрүп турган берилгендер.

Бул жерде берилгендер структурасы классификациялоо кандайдыр бир программалоо тилине байланышсыз жалпы назарияттык (теориялык) планда каралмакчы. Ал эми Turbo Pascal тилинин чөйрөсүндө тигил же бул берилгендер структурасы менен иш жүргүзүүгө болору же болбостугу өзгөчө белгиленет.

8.1. Статикалык структурадагы берилгендер

Статикалык структурадагы берилгендерди төмөнкүчө классификациялоого болот.

Жөнөкөй берилгендерди төмөндөгүчө классификациялоого болот.

Курама берилгендерди төмөндөгүчө классификациялоого болот.

Статикалык структурадагы берилгендер жөнөкөй структуралардан кандайдыр бир мыйзам боюнча куралып **жөнөкөй** (**скалярдык**) жана **курама** (**агрегативдик**) болушу мүмкүн. Программалоо тилдеринде жөнөкөй берилгендерге берилгендердин стандарттык типтери туура келип, аларга эреже катары, арифметикалык (натуралдык, бүтүн, чыныгы, комплекстик), символдук, бульдук жана көрсөткүчтүк (шилтемелик) типтерди кошушат. Turbo Pascal тилинде Byte, Word натуралдык типтери, Integer, Shortint, LongInt бүтүн типтери, Real, Single, Double,

Extended, Comp чыныгы типтери, Boolean, ByteBool, WordBool, LongBool бульдук типтери, Char символдук тиби жана Pointer көрсөткүчтүк типтери кошулган. Чыныгы типтер мында фиксирленген чекиттүү жана жылма ондук чекиттүү формада көрсөтүлүшү мүмкүн. Turbo Pascal да комплекстик сандарды көрсөтүү үчүн стандарттык типтер жок.

Андан сырткары, кээ бир программалоо тилдери (мындай алгачкы тил Pascal тили болгон) программистке өзүнүн менчик скалярдык типтерин, алар үчүн мүмкүн болгон маанилерди саноо жолу менен же башка скалярдык типтин маанилеринин камтылуучу диапазонун көрсөтүү менен баяндоого мүмкүнчүлүк берет. Turbo Pascal тилинде мындай мүмкүнчүлүктөр тиешелеш түрдө саналуучу жана интервалдык типтер көрүнүшүндө колдонулат.

Курама структурадагы берилгендер **бир тектүү** (бардык элементтери бир типте болгон) жана **бир тектүү эмес** (түрдү типтеги элементтер бир бүтүнгө бириктирилген) деп экиге бөлүнөт. Бир тектүү структурадагы элементтерге массивдер, жолчолор жана көптүктөр таандык, ал эми бир тектүү эмес структурадагы берилгендерге жөнөкөй жазуулар, варианттык жаазуулар, биригүүлөр жана объекттер кирет.

Массивдерди ар түрдүү эки белги боюнча классификациялоого болот:

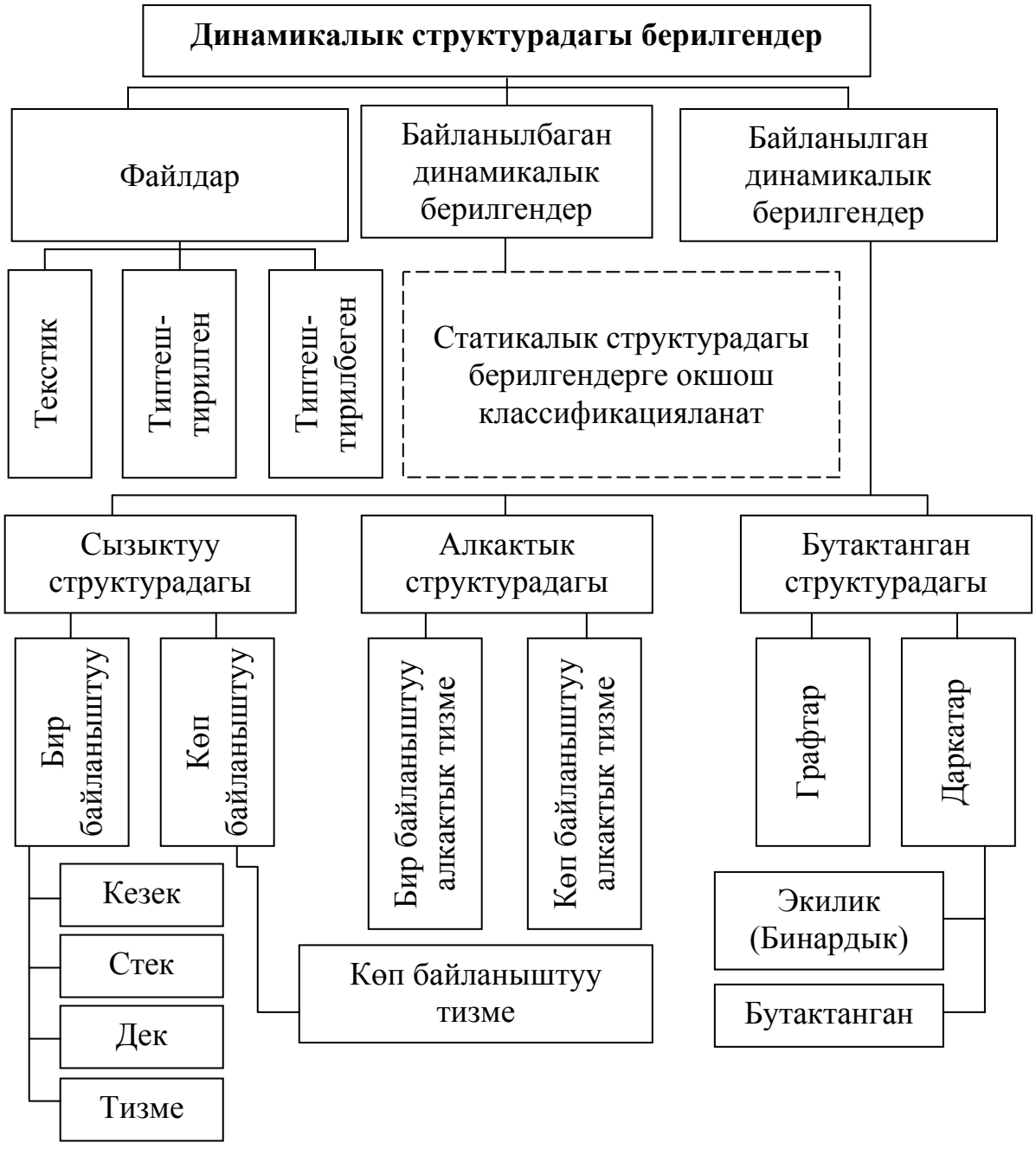
- өлчөмдөрү боюнча массивдер бир өлчөмдүү массивдер (векторлор), эки өлчөмдүү (матрицалар) жана көп өлчөмдүү (үч, төрт беш ж.б. өлчөмдүү) массивдер деп бөлүнөт;
- элементтеринин тиби боюнча массивдер жөнөкөй берилгендердин массиви, бир тектүү структуралардагы берилгендер массиви, бир тектүү эмес структурадагы берилгендер массиви жана файлдык массивдер деп бөлүнөт.

Turbo Pascal тилинде бир тектүү эмес структурадагы берилгендердин реализацияланышы боюнча мына муну айтууга болот. Жөнөкөй жана варианттык жазуулар өзүнчө синтаксистик конструкциялар аркылуу колдонулат.

Объекттер өзүнүн структурасы боюнча жазуларга окшош, бирок алардан айырмаланып жөнөкөй жана курама типтеги талааларды гана эмес процедуралык типтеги талааларды да өз ичине ала алат. Бул болсо берилгендердин структурасы катары объекттердин кубаттуулугун бир топ арттырат.

8.2. Динамикалык структурадагы берилгендер

Динамикалык структурадагы берилгендерге файлдар, байланган жана байланбаган динамикалык берилгендер кирет. Башка белгилүү болгон классификациялардан айырмаланып бул жерде файлдар динамикалык структурадагы берилгендерге киргизилди. Бул динамикалык берилгендердин жогоруда келтирилген аныктамасынан улам келип чыгат. Файлдын ортосуна элементтерди сыйлыгыштырууга (вставка) жана өчүрүүгү уруксат жок болгону менен программанын иштөө процесинде файлдын узундугу өзгөрүшү, чоңоюп же нөлгө чейин азайып кетиши мүмкүн. А бул болсо берилгендердин структурасы катары файлдын динамикалык касиети. Динамикалык структурадагы берилгендерди төмөнкүчө классификациялоого болот:



9. СТАТИКАЛЫК СТРУКТУРАДАГЫ БЕРИЛГЕНДЕР МЕНЕН ИШТӨӨ

Кеңешпей туруп жүзөгө ашырууга аракеттенилген иш ийгиликтүү бүтпөйт.
(Даанышман ойлор)

9.1. Жөнөкөй типтеги берилгендер менен иштөө

9.1.1. Жөнөкөй жана жолчолук типтеги өзгөрүлмөлөрдүн маанилерин кийрүү-чыгаруу

Turbo Pascal тилинде стандарттык кийрүү

Read (файлдык өзгөрүлмөнүн аты, өзгөрүлмөлөрдүн тизмеси),
жана
Readln (файлдык өзгөрүлмөнүн аты, өзгөрүлмөлөрдүн тизмеси),

тиркелген (встроенный) процедуралары менен, ал эми стандарттык чыгаруу

Write (файлдык өзгөрүлмөнүн аты, чыгаруу элементтеринин тизмеси),
жана
Writeln (файлдык өзгөрүлмөнүн аты, чыгаруу элементтеринин тизмеси)

процедуралары менен ишке ашырылат.

Стандарттык кийрүү Input атына ээ болгон алдын ала аныкталган, клавиатура менен байланышкан тексттик файлдан (из файла) аткарылат, ал эми стандарттык чыгаруу дисплей менен байланышкан Output атына ээ болгон алдын ала аныкталган тексттик файлына (в файл) аткарылат. Стандарттык кийрүү-чыгаруу учурунда файлдык өзгөрүлмөнүн атын Read, Readln, Write, Writeln процедураларында сөзсүз көрсөтүү шарт эмес, анткени көрсөтүлбөгөн учурда кийрүү үчүн Input аты, ал эми чыгаруу үчүн Output аты кабыл алынат. Ошентип, мына бул фрагменттер

```
Readln(Input, A, B);
Writeln(Output, 'A=', A, 'B=', B);
```

жана

```
Readln(A,B);
Writeln('A=', A, 'B=', B);
```

эквиваленттүү болушат.

Pascal тилинин бардык реализацияларында жана версияларында элементтеп (по элементный) кийрүү-чыгуу концепциясы пайдаланылат. Башкача айтканда **структураланган типтердеги өзгөрүлмөлөрдүн маанилерин жалгыз сандагы кийрүү (чыгаруу) процедурасы аркылуу кийрүүгө (чыгарууга) мүмкүн эмес**. **Мындай өзгөрүлмөлөрдү кийрүү (чыгаруу) үчүн цикл операторун пайдалануу зарыл**. Мындан четтөө String тибиндеги өзгөрүлмөлөр үчүн болот, себеби Tupbo Pascal тилинде бул тип "жолчолук тип" деп аталган атайын категорияга бөлүнгөн.

Стандарттык кийрүү-чыгаруу фрагменттерин жазууда төмөнкүлөрдү эсте тутуу зарыл:

- Read жана Readln процедуралары аркылуу **бүтүн, чыныгы, символдук жана жолчолук** типтердеги маанилерди гана окууга уруксат берилет.
- Write жана Writeln процедуралары аркылуу **бүтүн, чыныгы, символдук, жолчолук жана бульдук** типтеги гана маанилерди жазууга уруксат.

Жогоруда көрсөтүлгөн типтердин маанилерин Read жана Readln прцедураларынын жардамында кийрүү төмөнкү эрежелерди эсепке алуу менен аткарылат.

Кийрүү

1) Символдук типтеги өзгөрүлмөгө маанини кийрүү.

Программа	Баштапкы файл же клавиатура
var A , B , C : Char; begin Read (A); {A, $ га барабар болот} Read (B); {B, пробелге барабар болот} Read (C); {C, 7 ге барабар болот} end.	$ 7 ↑ └ пробел

Read процедурасы файлдан бир символду окуп, аны өзгөрүлмөгө ыйгарат. Эгерде Read процедурасынын аткарылыш алдында Eof функциясы (файлдын бүтүшү) True маанисин кабыл алган болсо, анда өзгөрүлмөгө ASCII коду 26 болгон символ ыйгарылат. Эгерде Eoln функциясы (жолчонун бүтүшү) True маанисине ээ болгон болсо, анда тиешелүү өзгөрүлмөгө ASCII коду 13 (каретканы кайтаруу символу) болгон символ ыйгарылат. Кийинки Read амалы файлдагы кийинки символдон башталат.

2) Бүтүн же чыныгы типтеги өзгөрүлмөгө маанини кийрүү.

Программа	Баштапкы файл же клавиатура
var A,B: Integer; X,Y: Real; begin Read(A); {A, 77ге барабар болот} Readln (B); {B, -34.24 кө барабар болот} Read(X); {X, 5.87e-5 ке барабар болот} Readln(Y); {Y, 555 ке барабар болот} end.	77 -34. 24 5.87e -5 555

Read процедурасы белгиге ээ болгон бүтүн же чыныгы сандарды түзүүчү символдордун удаалаштыгын күтөт. Сандар жолчосунан мурда келген ар кандай пробелдер, табуляция белгилери же жолчонун бүтүш белгиси каралбай өткөрүлүп жиберилет. Санды бөлүп алуу биринчи пробелди, табуляция символун, жолчонун же файлдын бүтүш белгисин кездештиргенге чейин аткарылат. Эгерде бөлүнүп алынган удаалаштык сандык форматка тиешелеш келбесе, анда кийрүү-чыгаруу катасы болуп өтөт. Кийинки Read процедурасынын аткарылышы алдынкы оператор келип токтогон пробелден, табуляция символунан же жолчонун бүтүш эн-белгисинен баштап аткарылат.

3) Маанини жолчо тибиндеги (String) өзгөрүлмөгө же нөл базалуу (array [0..N] of Char) символдор массивине кийрүү.

Программа	Баштапкы файл же клавиатура
{$X+} var A: array [0..5] of Char; S1, S2, S3: String [10]; begin Read(A); {A, ‘AAAAA’ га барабар болот} Readln (S1); {S1, ‘BBBBBBB’ га барабар болот} Read(S2); { S2 куру жолчого барабар болот} Readln; {жаңы жолчого өтүү} Read(S3); { S3, ‘ABCDEFGHIJ’ га барабар болот} end.	AAAAABBBBBBB ABCDEFGHIJ

Read процедурасы жолчолук өзгөрүлмөнүн жарыяланышында канча көрсөтүлгөн болсо баштапкы берилгендердин ошончо символун окуйт. Бул учурда пробел, сандык өзгөрүлмөгө (в числовую переменную) окугандагыдай, ажыраткыч болуп эсептелбейт. Эгерде баштапкы берилгендердин учурдагы жолчосунда өзгөрүлмөнүн баяндоосу боюнча талап кылынгандагыдан аз сандагы символдор калган болсо, анда бул жолчодогу бар болгон гана символдор окулуп чыгылат. Кийинки Read процедурасынын аткарылышы алдынкы жолчо аяктаган жолчонун бүтүш эн-белгисинен башталат. Башкача айтканда, **Read процедурасы жолчолук өзгөрүлмөнү окуганда жаңы жолчого өтүүнү жасабайт**. Мындай өтүү Readln процедурасы тарабынан гана аткарылышы мүмкүн.

Маанилерди нөл базалуу символдук массивдерге кийрүү үчүн Read жана Readln процедураларын пайдаланууга кеңейтирилген синтаксис ({$X+} директивасы) кошулган учурда гана уруксат берилет. Аяктоочу нөл-символ NULL (ASCII коду 0) автоматтык түрдө кошулат.

Readln процедурасынын ишинин Read процедурасынан айырмасы – ал Read процедурасынын аракеттери аткарылгандан кийин кийинки жолчого өтүүнү ишке ашырат жана параметрсиз Readln процедурасын чакыруу баштапкы берилгендерди окуунун учурдагы позициясын учурдагы жолчодо дагы берилгендердин бар же жок экендигинен көз карандысыз түрдө кийинки жолчонун башталышына которот.

Чыгаруу

Writeln процедурасы менен чыгаруунун ар бир элементи төмөнкү көрүнүшкө ээ.

```
Expr  [ :  MinField [ : DecDigits ] ]
```

мында:

Expr - тиби боюнча символдук, бүтүн, чыныгы, жолчолук же бульдук болушу мүмкүн болгон чыгарылуучу туюнтма;

MinField - чыгаруу талаасынын минималдык кеңдигин берүүчү, нөлдөн чоң болушу керек болгон бүтүн типтеги туюнтма;

DecDigits - ондук чекиттен кийин печатька чыгарылуучу ондук белгилердин санын берүүчү бүтүн типтеги туюнтма.

DecDigits бөлүгү, Expr чыныгы типке ээ болуп Minfield параметри көрсөтүлгөн учурда гана көрсөтүлүшү мүмкүн. Эгерде DecDigits параметри көрсөтүлсө анда сан фиксирленген чекиттүү форматта чыгарылат, а эгер көрсөтүлбөсө, анда жылма чекиттүү форматта чыгарылат.

Фиксирленген чекиттүү чыгаруу форматы:

[<пробелдер>] [-] <цифралар> [. <бөлчөк_бөлүгүнүн_белгилери>]

Жылма чекиттүү чыгаруу форматы:

[-] <цифра>.<бөлчөк_бөлүгүнүн_белгилери> E [+! -<даража_көрсөткүчү>]

Writeln процедурасы, Write процедурасынын аракеттеринен кийин жолчонун бүтүш белгисин жаза турган (“каретканы кайтаруу” жана “жолчону которуу” символдорун) Write процедурасынын кенейтирилиши болуп эсептелет.

Writeln процедурасын параметри жок чакырган кезде, файлга жолчонун бүтүш белгиси гана жазылат.

Мисал.

```
program TestWrite ;
    uses  Crt;
    const  i: Integer = 2 3456;
              r : Real = - 2345678;
              c:  Char = ’$’;
              b: Boolean = True;
              s:  String = ’Turbo Pascal’;
begin   ClrScr;
Writeln ( ‘Форматталбаган печать’);
Writeln ( i, r, c, b, s) ; Writeln;
Writeln (‘Форматталган печать’);
Writeln  (i :10, r :10 : 3,  c :10, b :10,  s :20 ); Writeln;
Writeln(‘Чын. сандарды фиксирленген форматта печаттоо’);
Writeln (r : 3: c );
Writeln (r : 5: 3);
Writeln (r : 10: 3);
Writeln (r : 11 : 7);
Writeln (r : 15 : 8);
Writeln (r : 25 : 8);
Writeln;
Writeln (‘ Чын. сандарды жылма форматта печаттоо’);
Writeln (r : 3);
Writeln (r : 5);
  Writeln (r : 11);
  Writeln (r : 17);
  Writeln (r : 25);
 end.
```

9.1.2. Жөнөкөй типтеги маанилер менен иштөө үчүн киркелген негизги процедуралар жана функциялар

Арфиметикалык типтер

Abs(x) функциясы – х аргументинин абсолюттук маанисин кайтарып берет.
Arctan(x) функциясы – х аргументинин арктангенсин кайтарып берет, мында х-радиандардагы бурч.
Cos(x) функциясы – х аргументинин косинусун кайтарып берет, мында х-радиандардагы бурч.
Sin(x) функциясы – х аргументинин синусун кайтарып берет, мында х-радиандардагы бурч .
Frac(x) функсиясы – х аргументинин бөлчөк бөлүгүнө барабар болгон санды кайтарып берет.
Int(x) фунциясы – х аргументинин бүтүн бөлүгүнө барабар болгон санды кайтарып берет.
Ln(x) функциясы – х аргументинин натуралдык логарифмин кайтарып берет.
Pi функциясы – pi санынын маанисин (3.1415926538979323 85) кайтарып берет.
Exp(x) функциясы – е санынын х-даражасына барабар болгон санды кайтарып берет.
Sqr(x) функциясы – х аргументинин квадратын кайтарып берет.
Sqrt(x) функциясы – х тен болгон квадраттык тамырды кайтарып берет.

Иреттик типтер

Dec(x,y) процедурасы – х санынын маанисин у ке азайтат. Бул процедура Dec(x) көрүнүшүндө колдонулушу мүмкүн, бул учурда х санынын мааниси 1 ге азаят.
Inc(x,y) процедурасы – х санынын маанисин у ке арттырат. Бул Ins(x) көрүнүшүндө колдонулушу мүмкүн, бул учурда х санынын мааниси 1 ге артат.
Odd(x) функциясы – х аргументи так болгон учурда True маанисин берет, б.а. аргументтин жуп же так экендигин текшерет.
Pred(x) функциясы – аргументтин учурдагы маанисинен алдын (мурда) келген маанисин кайтарып берет.
Succ(x) функциясы – аргументтин учурдагы маанисинен кийин келүүчү маанисин берет.

Көрсөткүчтүк типтер

Addr функциясы – берилген объекттин адресин кайтарып берет.
Ofs функциясы – берилген объект үчүн жылышты (смещение) кайтарып берет.
Ptr функциясы – базалык сегменттин адресин жана жылышты Pointer тибиндеги мааниге өзгөртүп түзөт.
Seg функциясы – берилген объект үчүн сегменттин адресин кайтарып берет.

9.1.3. Пайдалануучулук жөнөкөй типтер менен иштөө өзгөчөлүктөрү

Пайдалануучулук жөнөкөй типтерге саналуучу тип жана тип-диапазон кирет. Тип-диапазон тибиндеги өзгөрүлмөлөр менен иштөө аларга «ата-тектеш» типтер менен иштөө сыяктуу эле ишке ашырылат, бирок мүмкүн болгон маанилердин кичине диапазонун эске алуу менен. Программаларда саналуучу типтеги өзгөрүлмөлөрдү пайдалануу төмөнкүдөй өзгөчөлүктөр менен айкалышат:

- саналуучу типтер иреттик болуп эсептелет, ошондуктан алардын маанилерине иреттик тип үчүн уруксат берилген баардык процедуралар жана функциялар колдонумдуу;
- саналуучу типтердин маанилерин Read, Readln, Write, Writeln процедуралары үчүн аргумент катары пайдадалануруга болбойт;
- саналуучу типтеги операнддар үчүн пайдалануруга мүмкүн болгон амалдардын жалгыз тайпасы – бул катыш амалдарынын тайпасы. Кошуу, көбөйтүү, кемитүү дагы кандайдыр бир башка амалды саналуучу типтеги өзгөрүлмөлөр үчүн аткарууга уруксат берилбейт;
- саналуучу типтеги өзгөрүлмөлөрдү ыйгаруу операторлорунда пайдалануруга уруксат бар, ошондой эле аларды массивдердин индекстери жана for операторунун чек аралары катары пайдалануруга болот.

Саналуучу типтер үчүн уруксат берилген аракеттердин кээ бирөөлөрүн демонстрациялоочу мисалдарды келтиребиз.

Саналуучу типти жарыялоо:

```
Type Days = (Sunday,Monday,Tuesday,Wednesday, Thursday,
                                        Friday, Saturday);
     var   Yesterday,Today,Tomorrow :  Days;
```

Саналуучу типтеги өзгөрүлмөлөрдү жана маанилерди туюнтмаларда жана ыйгаруу операторлорунда пайдалануу:

```
if  Today=Saturday then
  begin
     Yesterday := Friday;
     Tomorrow := Sunday;
  end;
```

Саналуучу типтеги өзгөрүлмөлөрдү жана маанилерди тандоо операторунда пайдалануу:

```
case Today  of
   Sunday          :writeln('Бүгүн жекшемби');
   Monday         : writeln('Бүгүн дүйшөмбү');
   Tuesday         : writeln('Бүгүн шейшемби');
   Wednesday     : writeln('Бүгүн шаршемби');
   Thursday       : writeln('Бүгүн бейшемби');
   Friday           : writeln('Бүгүн жума');
   Saturday        : writeln('Бүгүн ишемби');
   end;
```

Саналуучу типтеги өзгөрүлмөнү жана маанилерди **for** циклдик операторунун спецификациясы катары пайдалануу:

```
for   Today  := Monday to Friday do
     bedin
       …
     end;
```

9.1.4. Типтерди өзгөртүп түзүүнүн стандарттык функциялары

Chr(x) функциясы - ASCII коду x бүтүн санына барабар болгон символду кайтарып берет. Мисалы Chr(65)='A'.

Ord(x) функциясы - саналуучу типтеги x маанисинин иреттик номерин кайтарып берет.

Round функциясы - чыныгы типтеги маанини, узун бүтүн типке ээ болгон мааниге чейин тегеректейт.

Trunc функциясы - чыныгы типтеги маанини, узун бүтүн типке ээ болгон мааниге чейин кесет.

9.2. Бир тектүү структурадагы курама берилгендер менен иштөө

9.2.1. Массивдер

Массив – бул номерлери боюнча иреттелген, жөнөкөй же курама структурадагы элементтердин бир тектүү, өлчөмү жана конфигурациясы боюнча фиксирленген жыйындысы болгон берилгендердин структурасы болуп эсептелет.

Массив анын аты (иденфикатору) жана массивдин талап кылынган элементинин жайгашкан ордун көрсөтүү үчүн зарыл болгон өлчөмдөрүнүн (координаталарынын) саны менен аныкталат. Массивдин аты анын бардык элменттери үчүн жалпы – бирөө гана болот.

Массивдин элементтеринин конфигурациясы фиксирленген (өзгөрбөс) болгондуктан, анын өзүнчө бир элементине, массивдин өлчөмүнөн көз каранды түрдө бир же бир нече индекстердин жардамында кайрылуу мүмкүн. Индекстер катары иреттик типтеги константаларды же өзгөрүлмөлөрдү пайдаланууга болот. Массивдин элементтери болуп каалаган типтеги жөнөкөй өзгөрүлмөлөр, ошондой эле курама типтеги өзгөрүлмөлөр (массивдер, жолчолор, жазуу ж.б. типтердеги) келиши мүмкүн.

Маселелерди чечүүдө, эреже катары бир өлчөмдүү, эки өлчөмдүү жана үч өлчөмдүү массивдер пайдаланылат. Андан чоң өлчөмдөрдөгү массивдер практикада сейрек кездешет.

Схемалык түрдө бир, эки жана үч өлчөмдүү массивдерди төмөндөгүчө көрсөтүүгө мүмкүн:

Бир өлчөмдүү массив (вектор)

Бир өлчөмдүү массивдин элементин Turbo Pascal тилинде массивдин атын, андан соң ага удаалаш чарчы кашаалардын ичинде анын индексин көрсөтүү менен жазууга болот. Мисалы A[5] – мында A – массивдин аты, 5 саны анын бешинчи элементинин индекси.

Бул массивди мындайча баяндоого болот:

```
const
  n =100;
  var
   A: array [1 .. n ]  of  Real;
```

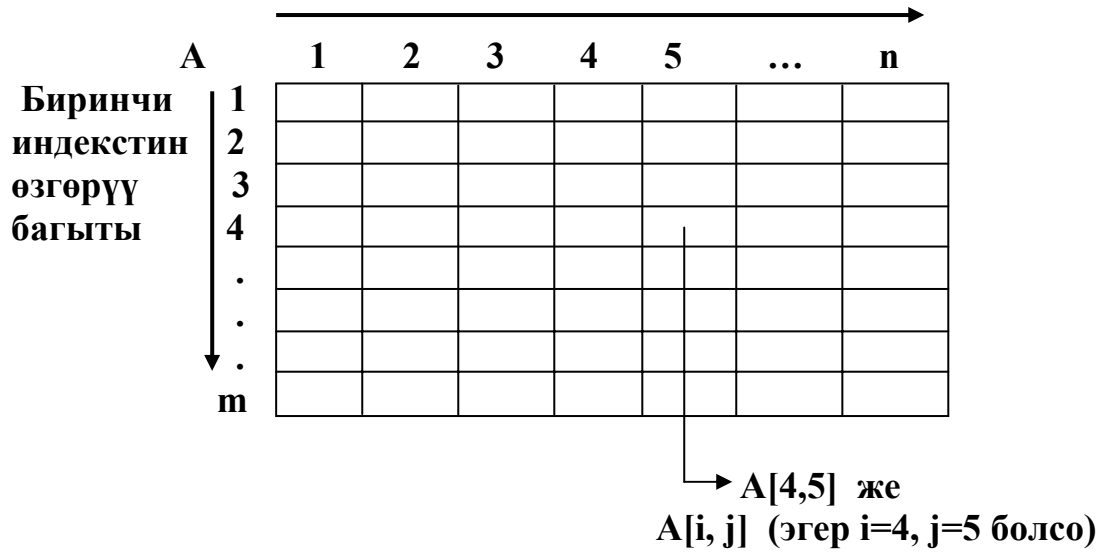
же

```
const n = 100;
    type
       T_Vector  = array [1 .. n] of  Real;
   var
       A: T_Vector;
```

Эки өлчөмдүү массив (матрица)

Turbo Pascal тилинде A атына ээ болгон эки өлчөмдүү массивдин (i,j) координаталарына ээ болгон элементи A[i , j] деп жазылат.

Экинчи индекстин өзгөрүү багыты



Бул массивди мындайча баяндоо мүмкүн:

```
const
   m =20 ;  n =30;
var
   A: array [1..m , 1..n] of Real;
```

же

```
const
    m=20; n=30;
type
     T_Matr = array [1..m , 1..n] of Real;
 var
    A:  T_Matr;
```

Үч өлчөмдүү массив

Үч өлчөмдүү массив да так эле эки өлчөмдүү массив сыяктуу жазылат. Мисалы, A[i, j, k] үч өлчөмдүү массивин Tupbo Pascal тилинде мындайча баяндоого болот. Мында i – бирнчи индекс, j- экинчи индекс ал эми k - үчүнчү индекс.

```
const
    m=10 ; n=40 ; p=30;
var
    A: array [1..m, 1..n, 1..p] of  Real;
```

же

```
const
    m = 10 ;  n = 40 ;     p = 30;
  type  T_Array = array [1..m, 1..n, 1..p] of Real;
  var
    A:  T_Array;
```

Массивдердин мүмкүнчүлүктөрүн демонстрациялоочу классикалык мисалдар болуп сорттоо жана издөө маселелери эсептелет.

9.2.2. Массивдерди сорттоо

Сорттоонун бир нече ар түрдүү алгоритмдери бар. Тигил же бул усул боюнча сорттоо маселесин чечүүдө адатта кошумча эсти минималдык пайдалануу талабы коюлуп, андан кошумча массивдерди колодонууга болбой тургандыгы келип чыгат.

Сорттоонун ар түрдүү усулдарынын алгоритмдеринин иштөө тездигин баалоодо, эреже катары, эки көрсөткүч пайдаланылат:

- ыйгаруулардын саны;
- салыштыруулардын саны.

Сортоонун бардык усулдарын эки чоң тайпага бөлүүгө болот:

- сортоонун түз усулдары;
- сортоонун жакшыртылган усулдары.

Принциби боюнча усулдун негизи болгон сорттоонун түз усулдарын өз кезегинде үч тайпачага ажырытууга болот:

1) сыйлыгыштыруу менен сорттоо (кошуу менен);
2) тандоо менен сорттоо (бөлүп алуу менен);
3) алмашуу менен сорттоо («көбүктүк» сорттоо).

Сорттоонун жакшыртылган усулдары түз усулдардын эле принцибине негизделет, бирок сорттоо процессин тездетүү үчүн кээ бир оригиналдуу идеяларды пайдаланат. Иштөө тездиги бир топ төмөн болгондуктан түз усулдар практикада сейрек колдонулат. Бирок, алар ушул усулдарга негизделген жакшыртылган усулдардын

маңызын жакшы көрсөтүп бере алат. Андан сырткары, кээ бир учурларда (массивдердин узундуктары анча чоң болбогон же/жана массивдин элементтеринин баштапкы жайланышы өзгөчө болгон учурда) түз усулдардын айрымдары жакшыртылган усулдардан да ашып түшүшү мүмкүн.

Сыйлыгыштыруу менен сорттоо

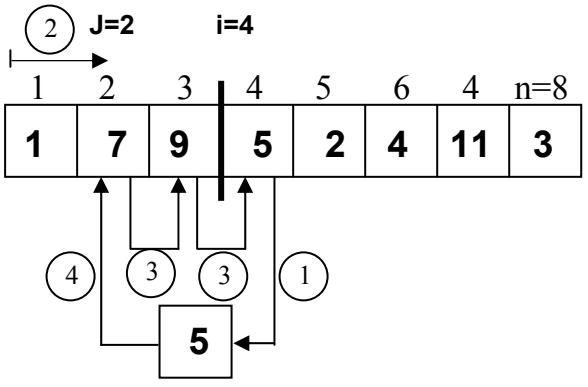
Усулдун иштөө принциби: Массив эки бөлүккө бөлүнөт: сорттолгон жана сорттолбогон. Сорттолбогон бөлүктөн элементтер кезек менен тандалып алынат жана сорттолгон бөлүккө андагы элементтердин иреттелиши бузулбай тургандай кылып сыйлыгыштырылып коюлат. Алгоритмдин ишинин башталышында массивдин сорттолгон бөлүгү катары биринчи элементти гана ал эми сорттолбогон бөлүк катары калган бардык элементтерди алышат.

Ошентип алгоритм ар бири төмөндөгүдөй төрт аракетти камтыган **n-1** (**n** - масиивдин өлчөмү) жүрүп чыгуудан (проход) турат:

- кезектеги сорттолбогон **i**-элементти алуу жана аны кошумча өзгөрүлмөгө сактоо;
- массивдин сорттолгон бөлүгүндө, алынган элементтин келиши элементтердин иреттелишин бузбай тургандай **j**-позицияны издөө;
- сыйлыгыштыруу үчүн табылган позицияны бошотуу үчүн массивдин элементтерин **i-1** - ден **j-1** - ге чейин оңго жылыштыруу;
- алынган элементти табылган **j**-позицияга сыйлыгыштырып коюу.

Ушул усулду реализациялоо үчүн, бири-биринен сыйлыгыштыруу үчүн позицияны издөө жолу менен айырмаланган бир нече алгоритмдерди сунуштоого болот. Ушундай алгоритмдердин бирөөсүн карап көрөбүз.

Схемалык түрдө бир жолку өтүүнүн баяндалган аракеттерин мындайча көрсөтүүгө болот:



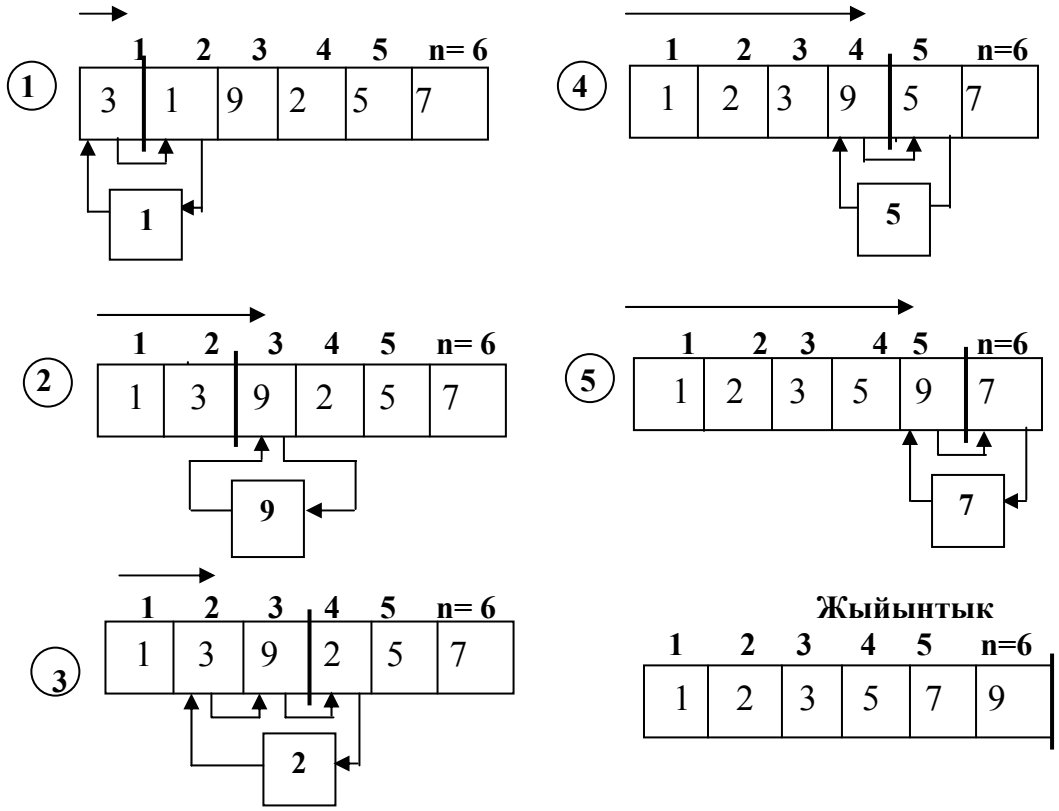
1 Сактоо

2 Сыйлыгыштыруу ордун издөө

3 Жылыш

4 Сыйлыгыштыруу

Сыйлыгыштыруу усулу менен сорттоонун схемасы. Сол жактагы тегерекчелерде өтүүнүн номери көрсөтүлгөн.



Бул каралган алгоритмди реализациялоочу программа төмөндөгүдөй көрүнүшкө ээ болот.

```
program  InsertionSort;
uses   Crt ;
const
     n=20 ; {массивдин узундугу}
type
    TVector = array [1 .. n] of Real;
var
Vector  : TVector;
  B       :  Real;
  i, j, k  : Integer;
begin
 ClrScr;
Writeln('Массивдин элементтерин киргизгиле:');
for i :=1 to n do  Read (Vector [ i ]); Readln;
{-------------------------------------------------------------------}
      for   i : = 1  to  n  do
  begin
         B  := Vector [ i ]: ;{Сорттолбогон элементти алуу}
         {Сыйлыгыштыруу позициясын издөө цикли}
       j:= 1;
```

```
   while ( B > Vector [ j ] )  do
     j := j+1;        {Цикл аяктагандан кийин j индекси          }
                      {сыйлыгыштыруу позициясын фиксирлейт}

{Сыйлыгыштыруу позициясын бошотуу үчүн}
{элементти жылыштыруу цикли}
   for k := i – 1 downto j do
        Vector [k+1] := Vector [k];
{Алынган элементти табылган позицияга коюу}
          Vector [j] := B;
end;
{-------------------------------------------------------------------------}
         Writeln  (′ Сорттолгон массив:′);
         for  i :=1 to n  do Write ( Vector [ i ] : 8 : 2);
        Writeln;
end.
```

Тандоо менен сорттоо

Усулдун принциби:

Массивде, **1**-элементтен **n**-элементке(акыркы) чейинки интервалда минималдык мааниге ээ болгон элементти таап (тандап алып) аны биринчи элемент менен орундарын алмаштырабыз. Экинчи кадамда **2**–элементтен **n**–элементке чейинки интервалда минималдык мааниге ээ болгон элементти таап аны экинчи элемент менен орундарын алмаштырабыз. Жана ушул сыяктуу бул процессти **n-1** – элементке чейин улантабыз.

Ушундай принцип менен иштөөчү массивди тандоо усулу менен сорттоо алгоритмин реализациялоочу программанын текстин келтирели.

```
program  SelectionSort;
uses   Crt;
const
     n = 20;       {массивдин узундугу}
type
    TVector = array [1 .. n] of Real;
var
    Vector : TVector;
    Min         : Real;
    Imin,  S   : Integer;
    i             : Integer;
begin  ClrScr;
   Writeln (′Массивдин элементтерин киргизгиле:′);
    for i :=1 to n do Read (Vector [i]); Readln;
```

```
   {-------------------------------------------------------------------}
       for   S := 1  to  n-1  do
    begin
{S-ден n-ге чейинки элементтердин диапазонунда}
{минималдык элементти издөө}
  Min :=Vector [S];
   Imin :=S;
   for  i := S+1 to n do
            if   Vector [i] < Min then
           begin
                  Min  := Vector [ i ];
                  Imin:= i;
            end;

{Минималдык жана S-элементтердин орундарын алмаштыруу}
             Vector [Imin] := Vector [S];
               Vector [S] := Min;
       end;

{-------------------------------------------------------------------}
          Writeln (′Сорттолгон массив:′);
             for  i := 1 to n Write(Vector [i] : 8 : 2);
          Writeln;
      end.
```

Алмашуу менен сорттоо

Бул усулду башкача дагы «көбүктүк» (“пузырьковая”) сорттоо деп аташат. Мындай аталыштын берилиши алгоритмдин аткарылышын образдык интерпретациялоодон келип чыгат. Чындыгында алгоритмдин аткарылыш процессинде бир кыйла “жеңил” элементтер аз-аздан “сырткы бетке” калкып чыгат.

Усулдун принциби:

Солдон оңду карай кезек менен коңушулаш эки элемент салыштырылат, эгер алардын өз ара жайланышы берилген иреттешкендик шартына тиешелеш келбесе, анда алардын орундары алмашылат. Андан ары кийинки эки коңушулаш элементтер каралат, ошентип бул процесс массивдин аягына чейин жүргүзүлөт.

Бир жолку мындай өтүүдөн (проход) кийин массивдин акыркы **n**-позициясында максималдык элемент туруп калат (демек биринчи “көбүкчө” “калкып” чыкты). Мындан кийин максималдык элемент өзүнүн акыркы позициясында туруп калгандыктан алмаштыруунун экинчи өтүүсү **n-1** – элементке чейин аткарылат. Ошентип отуруп бардыгы **n-1** өтүү талап кылынат.

Түз алмашуу менен сорттоо алгоритмин реализациялоочу программа төмөнкү көрүнүшкө ээ.

```
program  BubbleSort;
uses   Crt;
const
     n=20;   {Массивдин узундугу}
type
   TVector=array [1..n] of Real;
 var
Vector : TVector;
B        : Real;
i, k     : Integer;

  begin
    ClrScr;
Writeln (′Массивдин элементтерин киргизгиле:′);
for i := 1 to n do Read (Vector [i]) ; Readln;
   {-----------------------------------------------------------------}
    for k := n downto  2 do
{Кезектеги максималдык элементтин}
{ к- позицияга  “калкып чыгышы”}
      for i :=1 to k-1  do
         if   Vector [i]>Vector[i+1] then
       begin
         B := Vector [i];
       Vector [i] : = Vector[i+1];
       Vector[i+1] := B;
  end;

 {-----------------------------------------------------------------}
     Writeln (′ Сорттолгон массив:′);
     for  i :=1 to n do Write (Vector [1] : 8 : 2);
     Writeln:
 end.
```

Сорттоонун түз усулдарын салыштыруу

Сорттоонун түз усулдарынын алгоритмдерин салыштырып көрө турган болсок алар салыштыруулардын саны баюнча да, ыйгаруулардын саны боюнча да массивдин өлчөмү n-ден квадраттак көз карандылыкка ээ. Бул көз карандылыктан четтеген усул болуп **n∗ln(n)** тартибиндеги ыйгаруулар санына ээ болгон тандоо усулу эсептелет. Тандоо алгоритминин бул касиетин бир жолку салыштыруу үчүн көп сандагы ыйгаруулар аткарылуучу татаал структурадагы берилгендерди сорттоо маселелеринде колдонуу пайдалуу. Мындай маселелерде тандоо усулу сорттонун эң тез иштөөчү жакшыртылган усулдары менен конкуренцияга чыга алат.

Эгерде жогоруда карап өткөн түз усулдарды өз ара салыштырып көрө турган болсок, анда орто эсеп менен сыйлыгыштыруу жана тандоо усулдары дээрлик эквиваленттүү жана алмашуу усулуна караганда (массивдин узундугунан көз каранды түрдө) бир нече эсе жакшы.

9.2.3. Экилик издөө (бинардык издөө, жарымга бөлүп издөө)

Экилик издөө алгоритмин берилген элемнтти иреттелген массивдерде гана издеп табуу үчүн пайдалануу мүмкүн. Аны биз кемибеген тартипте (Vector [i]<= Vector [i+1]) иреттелген масивдин мисалында карайбыз.

Экилик издөөнүн принциби:

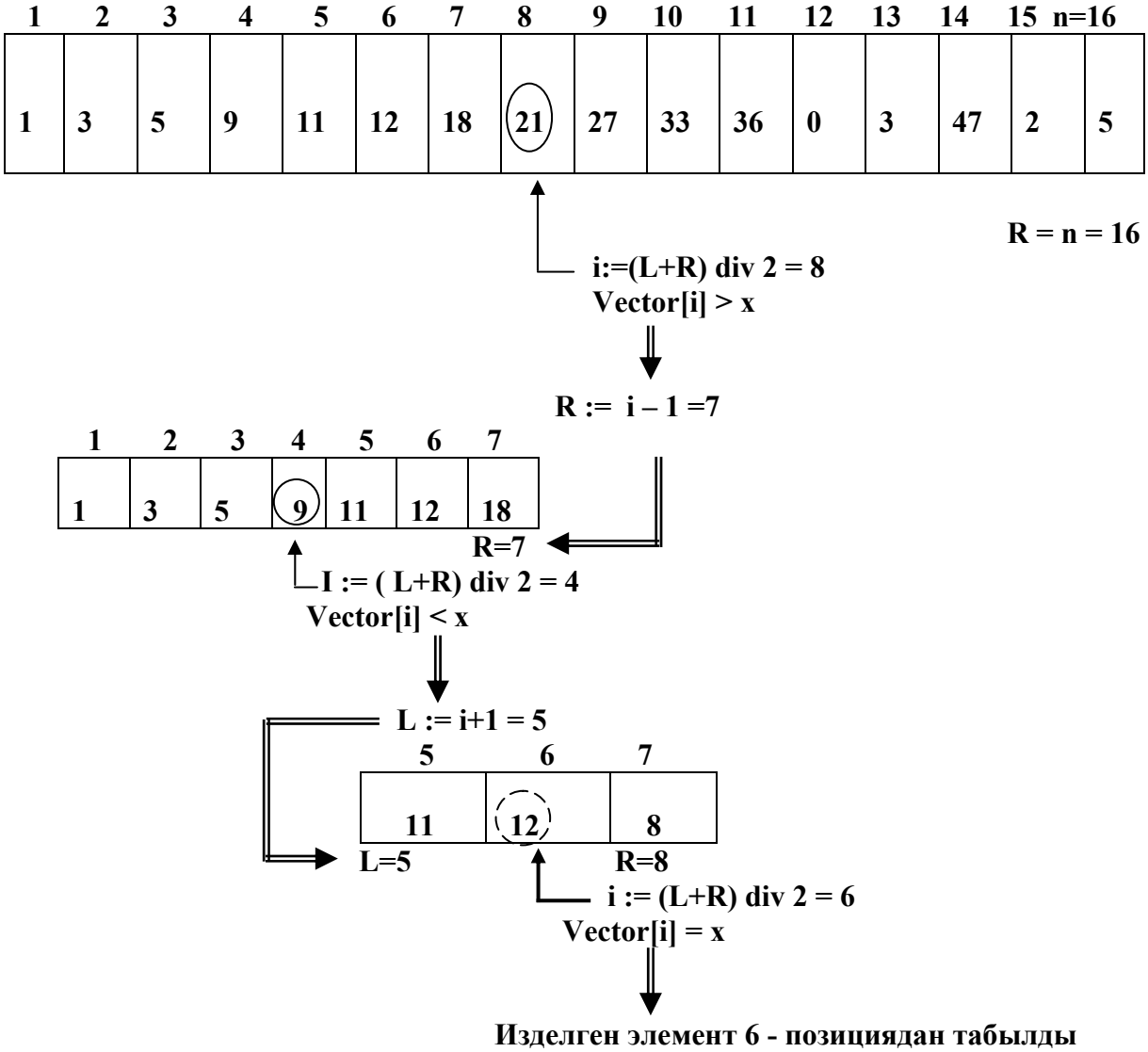
Берилген массив тең экиге бөлүнөт жана салыштыруу үчүн ортоңку элемент алынат. Эгерде ал изделип жаткан элемент менен дал келип калса, анда издөө бүтөт. Эгерде ортоңку элемент изделүүчү элементтен кичине болуп калса, анда анын сол жагындагы бардык элементтер изделген элементтен кичине болушат, ошондуктан алардын барысын андан аркы издөө зонасынан чыгарып салып, массивдин оң жаккы гана бөлүгүн калтырабыз. А эгер тескерисинче болсо б.а. ортоңку элемент изделгенден чоң болуп калса анда оң жаккы бөлүк таштalып, сол жаккы бөлүк калтырылат.

Экинчи этапта ушуга окшогон аракеттер калтырылган жарым бөлүктүн үстүнөн жүргүзүлөт. Натыйжада экинчи этаптан кийин массивдин 1/4 бөлүгү калат.

Ушундай кылып олтуруп бул процессти изделген элемент табылмайынча же издөө зонасынын узундугу нөлгө барабар болуп калмайынча уланта беребиз. Эгерде зонанын узундугу нөлгө барабар болуп калса анда изделген элемент табылбаган болот.

Экилик издөө алгоритминин схемасын карап көрөбүз.

Мында изделүүчү элемент x = 12
Баштапкы массив Vector



Экилик издөө усулун реализациялоочу программанын мүмкүн болгон варианттарынын бирөөсү төмөнкү көрүнүштө болот:

```
program  BinSeareh;
 uses   Crt;
  const
       n = 20; { Массивдин узундугу }
  type
       TVector = array [1 .. n]  of  Real;
  var
    Vector : TVector ; {Баштапкы массив}
     x         : Real ; { Изделүүчү элемент }
     L, R    : Integer ; { Издөө зонасынын учурдагы чектери}
      i         : Integer;
```

```
begin
   ClrScr;
   Writeln(′Массивдин элементтерин киргизгиле:′);
    for   i  := 1 to n do Read (Vector[ i ] ); Readln;
Write(′Изделүүчү элементти киргизгиле:′);
Readln(x);
 {-------------------------------------------------------------------------}
  L := 1 ;    R := n ;
  while  (L <=R ) do { Азырынча чектер кесилишпесе }
   begin
    i := (L+R) div2 ; {Ортоңку элементтин индекси}
      if  Vector[i] = x  then
         Break {издөө циклинен чыгуу       }
                {себеби элемент табылды}
            else
               if  Vector [i] < x  then  L := i+1
                                    else   R := i-1;
    end; {while}
    if   Vector [i] = x   then
        Writeln(′Изделген элемент′, i:3, ′позицияда табылды′)
        else
         Writeln(′Изделген элемент табылган жок′);
{-------------------------------------------------------------------------}
  end.
```

9.2.4. Эки өлчөмдүү массивдер менен иштөөгө мисалдар

Мисал1. Элементтери чыныгы сандар болгон **m**×**n** өлчөмүндөгү эки өлчөмдүү массив берилген. Анын бардык терс эмес элементтеринин маанилерин 1ге азайтып, ал эми терс элементтеринин маанилерин 1ге арттырып чыккыла.

```
program IncDecMatr;
  uses Crt;
      const     m=12;  { Жолчолордун саны }
                n =14;  { Мамычалардын саны}
  type
      TMatr = array [1..m , 1..n] of Real;
  var
      Matr  : TMatr;  { Баштапкы матрица}
      Finp  : Text;        {  Баштапкы берилгендердин файлы}
       i ,  j  : Integer;
  procedure  PrintMatr;
  var   i , j :  Integer;
```

```
begin
    for i :=1 to m do
      begin
            for j := 1 to n do  Write (Matr [i, j] : 8 : 2);
       Writeln;
       end;
    Writeln;
  end;
  begin
      ClrScr;
  {Матрицанын баштапкы маанилерин окуу}
      Assign(Finp, 'FINP.DAT' ) ; Reset ( Finp);
      for i:=1 to m do
      begin
          for j :=1 to n do  Read  (Finp, Matr[i, j]);
            Readln (Finp);
      end;
    {- - - - - - -------- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
      Writeln ( 'Баштапкы матрица');
      PrintMatr;
  {Терс эмес элементтердин маанилерин 1ге азайтуу, ал эми }
        {терс элементтердин маанилерин 1ге арттыруу}
        for i:=1 to m do
        begin
           for j:=1 to n do
                  if Matr[i,j]>=0
                        then Matr [i, j] := Dec (Matr[i, j])
                        else Matr [i, j] := Ins ( Matr[i, j]);
        end;
    {- ---------- - - -- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
    Writeln('Өзгөргөн матрица:');
     PrintMatr;
  end.
```

Мисал 2. Элементтери бүтүн сандар болгон **m**×**n** өлчөмүндөгү эки өлчөмдүү массив берилген. Вертикалдык симметрия огуна карата матрицанын элементтеринин күзгүлүк (зеркальный) чагылышын (биринчи мамычанын элементтарин акыркы мамычаныкы менен алмаштыргыла, жана экинчи мамычанын акыркыдан мурунку мамычаныкы менен д.у.с.) жасагыла.

```
program VertMirrow ;
  uses  Crt ;
   const   m =15 ;  { Жолчолордун саны }
             n = 20 ; { Мамычалардын саны }
```

```
  type
    TMatr = array[1..m, 1..n] of Integer;
var
     Matr: TMatr; {Баштапкы матрица}
     Finp: Text;      {Баштапкы берилгендер файлы}
     B    :  Integer;
     i, j  :  Integer;
    procedure   PrintMatr;
   var i, j : Integer;
   begin
     for i := 1 to m do
        begin
          for j := 1 to n do Write (Matr [i, j]:5);
         Writeln;
       end;
    begin
        ClrScr;
      { Матрицанын баштапкы маанилерин окуу}
      Assign (Finp, 'FINP. DAT'); Reset (Finp);
       for i :=1 to m do
      begin
          for j :=1 to n do Read (Finp, Matr [i, j]);
           Readln ( Finp);
      end;
{- - - - --------- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
     Writeln ('Баштапкы матрица:');
      PrintMatr;
{ - - - - - - - ----- - - - - - - - - - - - - - ------ - - - - - - - - - - - - - - - }
{Вертикалдык симметрия огуна карата матрицанын }
{«күзгүлүк чагылышы»}
   for j :=1 to n div 2  do       {1-ден ортодогуга чейинки }
                                     {мамычаларды алабыз       }
   for I :=1 to m do     {Симметриялуу мамычалардын }
                            {оорундарын алмаштырабыз     }
   begin
         B  := Matr[i, j];
        Matr[i, j] := Matr [i, n-j+1];
   Matr[i, n-j+1]:= B
   end;
   {------------ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
    Writeln ('Өзгөртүп түзүлгөн матрица:');
     PrintMatr;
  end.
```

9.2.5. Жолчолор

Жолчо – бул символдордун бир өлчөмдүү массивинин орчундуу айырмачылыкка ээ болгон өзгөчө формасы болуп саналат. Символдордун массиви аны баяндоо кезинде аныкталуучу фиксирленген узундукка (элементтердин санына) ээ. Жолчо узундуктун эки түрүнө ээ болот:

- жолчонун жалпы узундугу, ал жолчону баяндоо кезинде бөлүнүп коюлуучу эстин өлчөмүн мүнөздөйт;
- жолчонун учурдагы узундугу - (дайыма жалпы узундуктан кичине же ага барабар) убакыттын ар бир конкреттүү моментинде жолчолордун маани берүүчү (смысловый) символдорунун санын көрсөтөт.

Программалоо тилдеринде жолчолорду реализациялоонун эки жолу (ыкмасы) пайдаланылат. Ошолорду өз-өзүнчө карайлы.

Жолчолорду реализациялоонун биринчи ыкмасы

Реализациялоонун биринчи ыкмасында, жолчонун учурдагы узундугу анын нөлдүк (б.а. 0 индексине ээ болгон) элементинде көрсөтүлөт. Бул элементке коду учурдагы узундуктун маанисине барабар болгон символ жазылат. Жолчонун нөлдүк элементи бул учурда пайдалануучу үчүн көрүнбөс кылынган, бирок аны программаларда пайдалануу сунуш кылынбаганы менен негизинен пайдаланса болот.

Ар бир символ эстин бир байтын ээлегендиктен учурдагы узундукту көрсөтүүнүн мындай жолунда жолчонун максималдык мүмкүн болгон узундугу, эстин бир байтына жазууга мүмкүн болгон максималдык маани менен чектелген болот. Башка сөз менен айтканда жолчонун учурдагы максималдык узундугу 255 символдон узун эмес болушу мүмкүн.

Учурдагы узундукту көрсөтүүнүн мындайча ыкмасы биринчи жолу Turbo Pascal тилинде алдын ала аныкталган string тиби көрүнүшүндө киргизилген, ушул себептен ушундай принцип боюнча реализацияланган жолчолорду көбүнчө Turbo – жолчолор же String – жолчолор деп аташат. Жолчолорду көрсөтүүдөгү мындай ыкманын (жолдун) артыкчылыгы - жолчолор менен болгон иш-аракеттерди эффективдүү аткарууга мүмкүнчүлүк берүүчү жолчонун учурдагы узундугуна болгон кайрылуунун өтө жөнөкөй болушу. Жетишпестиги – жолчонун максималдык узундугуна болгон чектөө.

коду 7 ге барабар болуучу символ.
n - жолчонун фиксирленген жалпы узундугу;
p - жолчонун учурдагы узундугу

Turbo Pascal тилинин 7.0 версиясында анын мурдагы баардык версияларындагыдай эле, жолчолорду реализациялоонун биз карап өткөн ыкмасына String алдын ала аныкталган тиби тиешелеш коюлган. Баяндоо кезинде аныкталуучу жолчонун жалпы узундугу чарчы кашаалардын ичинде көрсөтүлөт.

Мисалы,

```
var
   Str1:  String [12];
   Str2:  String [136];
   SMax: String;
```

Эгерде баяндоо кезинде узундук көрсөтүлбөсө, анда көрсөтүлбөгөн учурда (по умолчанию) 255 символго барабар болгон максималдык узундук кабыл алынат.

Srting тибиндеги жолчолор менен иштөө үчүн Turbo Pascal тилинде төмөндөгүдөй процедуралар жана функциялар пайдаланылат:

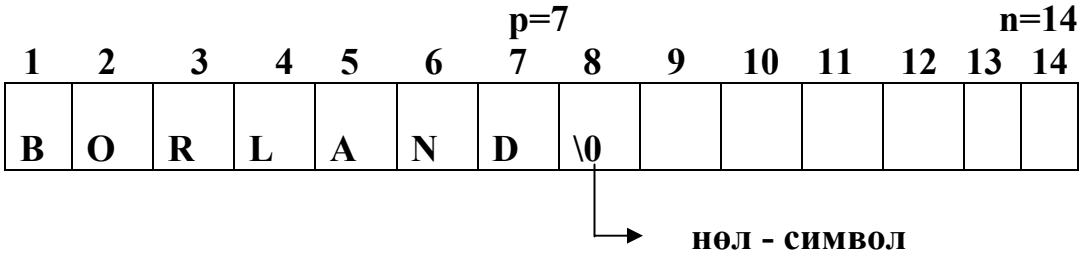
Concat функциясы – жолчолор удаалаштыктарынын конкатенация-сын(уламалаштырууну) аткарат.
Copy функциясы – берилген жолчодон андагы камтылма жолчонун көчүрмөсүн кайтарып берет.
Length функциясы – берилген жолчонун учурдагы (иш жүзүндөгү) узундугун берет.
Pos функциясы – жолчодо, камтылма жолчону издейт.
Delete процедурасы – жолчодон, камтылма жолчону жоготот (өчүрөт).
Insert процедурасы – жолчого камтылма жолчону сыйлыгыштырат.
Str процедурасы – сандык маанини, анын жолчолук көрсөтүлүшүнө өзгөртүп түзөт.
Val процедурасы – жолчолук маанини, анын сандык көрсөтүлүшүнө өзгөртүп түзөт.

Бул функциялар жана процедуралар үчүн эске тута турган нерсе: эгер жыйынтык болуп эсептелген жолчонун узундугу 255 тен ашып кете турган болсо, анда 255 тен ашкан символдордун удаалаштыгы кесилип ташталат.

Жолчолорду реализациялоонун экинчи ыкмасы

Жолчолорду реализациялоонун экинчи ыкмасында жолчонун учурдагы узундугу анын акыркы маани берүүчү символунан кийин коюлуучу атайын символ – белги менен фиксирленет. Turbo Pascal тилинде мындай белги катары “нөл-символ” деп аталуучу жана NULL деп же 0 деп белгиленүүчү нөл коддуу символ пайдаланылат. Учурдагы узундукту көрсөтүүнүн мындай ыкмасы практикалык жактан мүмкүн болгон максималдык узундукту чектебейт. Бирок, IBM – биргелешкен компютерде мындай чектөө бар, ал эстин бир сегменти (65 534 байт) өлчөмүндө коюлган.

Нөл менен аяктоочу жолчолорду дагы башкача ASCIIZ – жолчолор деп аташат.



Нөл менен аяктоочу жолчолор, нөл базалуу (б.а. төмөнкү чеги нөлгө барабар болгон) символдук массивдер көрүнүшүндө сакталат.

```
const
     Len = 411;
 type
   TStr9 = array [0 . . 8]  of  Char;
   TStr80 = array [0 . . 80] of Char;
   TstrLen = array [0 . . Len] of Char;
```

Андан сырткары Turbo Pascal тилиндеги PChar алдын ала аныкталган тиби бар. Ал нөл менен аяктоочу жолчодо көрсөткүч болуп эсептелет жана төмөнкүчө аныкталат:

```
type
   Pchar =  ^Char
```

Кеңейтирилген синтаксистик {$x+} директивасы коюлган учурда (ал көрсөтүлбөгөн учурда (по умолчанию) кабыл алынат) нөл базалуу символдук массив PChar тиби менен биргелешкен болот. Мындан сырткары, жолчолук константалар дагы ыйгаруу боюнча PChar тиби менен биргелешкен.

Мисалы;

```
uses Crt;
   const
      Str80 :  array [0 . . 80] of  Char= ‘PChar  тибиндеги жолчо’;
```

```
var
    P := Pchar;
begin
     ClrScr;
      P := ‘Pchar тибиндеги жолчоң’;
      Writeln (P);
      P := Str80;
      Writeln (P);
end.
```

Анан дагы, эгерде тиешелүү формалдык параметрлер PChar тибине ээ болсо, анда жолчолук константаларды иш жүзүндөгү (фактический) параметрлер катары берүүгө болот.

```
const
    Name:  Pchar = ‘Tutbo Pascal’;
procedure Test (Str :  PChar);
  begin
   . . .
  end;
 begin
    . . .
    Test (Name);
    . . .
  end.
```

PChar тибиндеги типтештирилген константаларды колдонууга уруксат берилет, бирок алардын узундуктары дагы String тибиндеги типтештирилген константалар сыяктуу 255 символ менен чектелген.

Тиби PChar болгон өзгөрүлмөлөрдү нөл базалуу массивге окшош индекстөөгө болот.

```
const
     Str 80 : array[0..80] of char= `Turbo Pascal`;
 var   P :  PChar;
begin
    P := Str80;
  Writeln ( ‘str 80 [5]=’, str 80 [5]); {Бул эки оператор  тең бир }
                                        {эле `n` символун печаттайт}
  writeln ( ‘p[5]=’, p[5]);
end.
```

Turbo Pascal тили {$X+} директивасы коюлган учурда PChar тибиндеги көрсөткүчтөр менен иштөө үчүн кошумча амалдардан пайдаланууга мүмкүнчүлүк берет.

Суммалоо (+) жана кемитүү (-) амалдары көрсөткүчтүн жылышын (смещение) берилген чоңдукка чоңойтуу жана азайтуу үчүн пайдаланылат. Андан сырткары кемитүү амалын PChar тибиндеги эки көрсөткүчтүн жылыштарынын айырмасын эсептөө үчүн пайдаланууга болот.

Мисалы,

```
var
     Str1,  Str2,  Res : PChar;
     i, j : Word;
 begin
  . . .
  Res := Str1+ I;    {Res , Str1 ге караганда  i  сисволго}
                     { узагыраак (алысыраак)  көрсөтөт}
  Res := Str2- I;    { Res, Str2 ге караганда  I  символго}
                     {жакыныраак көрсөтөт                }
   j := Str1 - Str2; { j, Str1 жана Str2 лер көрсөткөн символдор-}
                     {дун арасындагы символдордун санына      }
                     {барабар болот.}
                     {Str1, Str2 экөө тең бир эле символдук    }
                     {массивди көрсөтүүлөрү керек              }
 end.
```

Turbo Pascal тилинин стандарттык синтаксиси көрсөткүчтөрдү салыштырууда, алардын барабардыгын же барабарсыздыгын гана аныктоого мүмкүнчүлүк берет. Кеңейтирилген синтаксис болсо ({$x+} директивасы коюлган кезде) Pchar тибиндеги маанилерге <, >, <= жана >= амалдарын да колдонууга уруксат берет. Бирок, салыштырылып жаткан эки көрсөткүч бир эле символдор массивин көрсөтүп турат деген шартта.

Нөл менен аяктоочу жолчолор менен болгон амалдарды колдоо (подержка) үчүн Turbo Pascal тилинин комплектине жаңы Strings стандарттык модулу кошулган.

Ушул strings модулунун функцияларынын кыскача баяндамаларын келтиребиз.

StrCat функциясы – бир жолчону экинчи жолчонун аягына кошот жана көрсөткүчтү жыйынтыктоочу жолчого кайтарат.

StrComp функциясы – эки s1 жана s2 жолчолорун салыштырат. Эгер s1<s2 болсо, анда жыйынтык терс сан болот; s1=s2 болсо 0гө барабар болгон сан болот, эгер s1>s2 болсо, анда жыйынтык оң сан болот.

StrCopy функциясы – бир жолчонун маанисин экинчи жолчого көчүрөт.

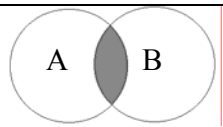
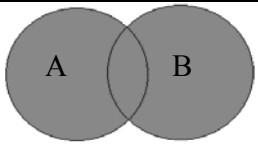
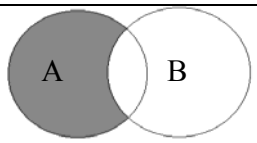
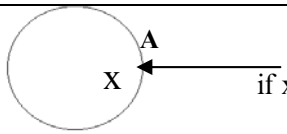
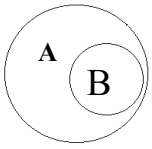
Көрсөткүчтү жыйынтыктоочу жолчонун башталышына кайтарат.

StrDispose	функциясы – мурда StrNew функциясынын жардамында бөлүштүрүлгөн жолчону жок кылат.
StrEcopy	функциясы – бир жолчонун маанисин экинчи жолчого көчүрөт. Көрсөткүчтүү жыйынтыктоочу жолчонун бүтүшүнө кайтарат
StrEnd	функциясы - көрсөткүчтү жолчону аяктоочу нөлдүк символго кайтарат.
StrIComp	функциясы - StrComp сыяктуу эки жолчону салыштырат, бирок символдордун регистрлерин айырмалабайт.
StrLCat	функциясы – баштапкы жолчону багытталган (целевой) жолчонун аягына бириктирет. Мында жыйынтыктоочу жолчонун узундугу берилген максимумдан ашпашы камсыз кылынат. Көрсөткүч жыйынтык- жолчого кайтарылат.
StrLComp	функциясы – берилген максималдык узундуктагы эки жолчону салыштырат.
StrLCopy	функциясы – баштапкы жолчодон берилген сандагы символдорду жыйынтыктоочу жолчого көчүрөт жана көрсөткүчтү жыйынтыктоочу жолчого кайтарат.
StrLen	функциясы - жолчонун узундугун кайтарат.
StrLIComp	функциясы - берилген максималдык узундуктагы эки жолчону сиволдор регистрин айырмалабастан салыштырат.
StrLower	функциясы - жолчону төмөнкү регистрге өзгөртүп түзөт жана ага көрсөткүчтү кайтарат.
StrMove	функциясы - баштапкы жолчодон символдордун удалаштыгын багытталган жолчого которот(жылдырат), көрсөткүчтү багытталган жолчого кайтарат.
StrNew	функциясы - динамикалык бөлүштүрүлүүчү областта жолчо үчүн эсти бөлөт.
StrPas	функциясы - нөл менен аяктоочу жолчону String-жолчого өзгөртүп түзөт.
StrPCopy	функциясы - String-жолчону нөл менен аяктоочу жолчого көчүрөт жана көрсөткүчтү нөл менен аяктоочу жолчого кайтарат.
StrPos	функциясы - көрсөткүчтү берилген камтылта жолчонун, жолчого биринчи киришине(первое вхождение) кайтарат, же эгер изделген камтылма жолчо жолчодо кармалып турбаса анда **nil** ге кайтарат.
StrRScan	функциясы - көрсөткүчтү көрсөтүлгөн символдун жолчодогу акыркы кирүүсүнө которот, же эгер символ жолчодо жок болсо анда **nil** ге которот.
StrScan	функциясы - көрсөткүчтү көрсөтүлгөн символдун жолчодогу биринчи кирүүсүнө которот, же эгер символ жолчодо жок болсо анда **nil** ге которот.
SteUpper	функциясы - жолчону жогорку регистрге өзгөртүп түзөт жана ага көрсөткүчтү кайтарат.

9.2.6. Көптүктөр

Программалоодо “көптүктөр” термини аны математикалык түшүнүүгө окшош пайдаланылат. Болгон айырмасы, Turbo Pascal тилинде көптүктөр иреттик типтеги элементтерди гана өз ичине алып тура алат. Кандайдыр бир конкреттүү көптүктүн (өзгөрүлмөнүн же типтештирилген константанын) элементтери, **базалык тип** деп аталган бир типке таандык болушу керек. Көптүктүн базалык тибинин маанилеринин максималдык саны анын **кубаттуулугу** деп аталат. Turbo Pascal тилинде базалык типтер катары кубаттуулугу 256 мааниден ашпаган иреттик типтерди пайдаланууга болот. Андан сырткары, базалык типтин төмөнкү жана жогорку чектеринин иреттик маанилери 0 дон 255 ке чейинки диапазондун пределинен чыгып кетпеши керек. Ошондуктан көптүктөрдүн базалык типтери катары Shortint, Integer, Longint, Word типтерин пайдаланууга болбойт.

Көптүктөр менен иштөө үчүн уруксат берилген амалдар жана алардын операндларынын типтери 4-бапта каралган. Материалды жакшы түшүнүү үчүн ар бир амалдын графиктик баяндоосунан турган жадыбалды келтиребиз.

Математи-калык белгилениш	**BP да белгиле-ниши**	**Амал**	**Мисал**
$\cap$	*	**Кесилишүү**	 C=A ∩B C:=A*B Жыйынтыктын тиби-көптүк
$\cup$	+	**Биригүү**	 C=A∪B C:=A+B Жыйынтыктын тиби-көптүк
\	-	**Айырма**	 **C=A\B** C:=A-B Жыйынтыктын тиби-көптүк
$\in$	**in**	**Камтылуучу лук (элементтин көптүккө)**	 **Жыйынтыктын тиби- boolean**
$\subset (\subseteq)$ $\supset (\supseteq)$	<= >=	**Камтылуучу көптүк болот** **Камтылуучу көптүктү өз ичине алат**	***Операндар-көптүктөр***  A⊃B (B⊂A) A>=B (B<=A) **Жыйынтыктын тиби - boolean**

Көптүктөр көптөгөн маселерди чечүү үчүн ийкемдүү жана көрсөтмөлүү механизм болуп саналат. Айрым учур катары алар жолчолор жана тексттер менен иштөөдө пайдаланылат. Мисал катары төмөнкү маселени карайбыз.

Маселе. Текстик файлда цифралардын жана латын тамгаларынын санын өз-өзүнчө эсептегиле.

```
program TestSets;
  uses Crt;
  type
     CharSet =  set  of  Char;
  const
     Digits:   CharSet  =  [ '0' . . '9' ];
    Letters:   CharSet = [ 'a' . .' z', 'A' . . 'Z' ];
  var
    CountDig,  CountLet :   Word;
     Finp :  Text;
    Ch:  Char;
 begin
    ClrScr;
    Assign  (Finp, 'TESTSETS.DAT');
    Reset ( Finp );
   While  not  eof  ( Finp )   do
   begin
     Read  (Finp , Ch);
      if  Ch in Digits then CountDig : = CountDig + 1;
      if  Ch in Letters the CountLet  : = CountLet + 1;
 end;
 Writeln(' Цифралардын саны =', CountDig : 5 );
 Writeln(' Тамгалардын саны =', CountLet :5);
end.
```

Программада пайдаланылган Letters көптүгү зарыл болгон учурда башка эки көптүктөн

```
SmallLetters = [ 'a' . . 'z' ];
BigLetters      = [ 'A' . . 'Z' ];
```

кошуу (биригүү) амалы -

```
Letters := SmallLetters +BigLetters;
```

аркылуу алынышы мүмкүн.

Ошондой эле тескери амалды жүргүзүү мүмкүн: Letters көптүгүнөн көптүктөрдү кемитүү амалынын жардамында SmallLetters жана BigLetters көптүктөрүн алууга болот.

```
SmallLetters  =  Letters - [ 'A' . . 'Z' ];
BigLetters      =  Letters - [ 'a' . . 'z' ];
```

9.3. Бир тектүү эмес структурадагы курама берилгендер менен иштөө

9.3.1. Жазуулар

Жазуу – бул массивдерден, көптүктөрдөн жана файлдардан айырмаланып берилгендердин курама структурасы болуп эсептелет.

Эгер өзүнчө алынган массив, көптүк же файл дайыма бирдей типтеги элементтерден турса, анда жазуу бишка типтеги берилгендердин каалагандай сандагы структураларын бир бүтүнгө бириктирет.

Turbo Pascal тилиндеги эки түрдүү - фиксирленген (кадимки) жана варианттык жазууларды айырмалап карашат.

Фиксирленген жазуулар

Кадимки фиксирленген жазуу бир же бир нече талаалардан туруп алардын ар бири үчүн жарыялоо кезинде аты (идентификатор) жана тиби көрсөтүлөт. Мисал катары студенттин жетишүүсүнүн өздүк карточкасын баяндоочу жазууну келтиребиз.

Мисал.

```
type  String8  =  String [8];
      String22  =  String [22];
      TStudentCard  =  record
         SurName  :  String22;        { Теги}
         Name  :  String22;           {Аты}
         FatherName:  String22;       {Атасынын аты}
         Year:  Integer;              {Туулган жылы}
         HomeAddress:  String;        {Үй адреси}
         GtoupCode:  string7;         { Тайпасынын шифри}
         MathAnal: Byte;              {Матем. анализ}
         SyzAl:  Byte;                {Сызык. алгебра}
         Prog:    Byte;               {Программалоо}
         Phys:  Byte;                 {Физика}
  end;
```

Бул келтирилген мисалда бирдей маанидеги жүктү алып жүргөн (MathAnal, SyzAl, Prog, Phyz) талаалардын группасы бар экендигин байкоо кыйын эмес. Программанын окумдуулугун (читабельность) көтөрүү жана группалык амалдардын аткарылышынын ыңгайлуу болушу көз карашы менен алганда мындай талааларды “жазуу” тибиндеги берилгендердин өзүнчө структурасына бириктирип алуу максатка ылайыктуу. Натыйжада TStudentCard тиби төмөнкүдөй көрүнүшкө ээ болот:

```
type  TStudentCard  =  record
        SurName : String22;          {Фамилиясы}
        Name  :  String22;           {Аты}
        FatheName : String22;        {Атасынын аты}
        Year    : Integer ;          {Туулган жылы}
        Homeaddres: String;          {Үй адреси}
        GroupCode:  String7;         {Группанын шифри}
       Marks : record
           MathAnal  :  Byte;        {Мат.анализ}
           SyzAl   : Byte;           {Сыз. алгебра}
           Prog    : Byte;           {Программалоо}
           Phys  :   Byte;           {Физика}
         end;
      end;
```

Дагы да жакшыраак мындайча жазууга болот:

```
type
     TMarks  =  record
     MathAnal  :  Byte;          {Мат.анал}
     SyzAl         :  Byte;      {Сыз. алгебра}
     Prog  :  Byte;              {Программалоо}
     Phys  :  Byte;              {Физика}
   end;
   TStudentCard  =  record
    SurName  :  String22;        {фамилиясы}
    Name       :  String22 ;     {Аты}
    FatherName  :  String22 ;    {Атасынын аты}
    Year          :  Integer;    {Туулган жылы}
    HomeAddress  :  String;      {Үй адреси}
    GroupCode  :   String7;      {группасынын шифри}
    Marks   :   Tmarks;          {Акыркы семместрдеги баалар}
  end;
```

Жазуунун талааларына кайрылуу квалификацияланган (такталган) идентификаторлордун жардамында, аларда «жазуу» тибиндеги өзгөрүлмөнүн идентификаторунан баштап талап кылынган талаанын атына чейинки бүтүндөй чынжырчаны көрсөтүү менен аткарылат. Квалификацияланган идентификаторлордун талааларынын аттары чекиттер менен ажыратылат. Мейли төмөндөгүдөй тип жана өзгөрүлмөлөр баяндалган болсун.

```
type
     TGroup = array [1 . . 25] of  TStudentCard;
   var
     Group_KB51,
     Group_KB52 : TGroup;
```

Бул учурда төмөндөгүдөй операторлор жана жазуунун талааларына кайрылуулар корректтүү болот.

```
Group_KB51 [1] . Name   : = ‘Nazgul’;
Group_KB51 [1] . Year   : =  1990;
Group_KB51 [1] . Marks.Prog   : = 5;
Group_KB51 [1] . Marks   : = Group_KB52  [1] . Marks;
Group_KB51 [5]           : = Group_KB52  [7];
Group_KB51               : = Group_KB52;
```

Жазуулар менен болгон ишти жөнөкөйлөтүү жана программага жакшы көрсөтмөлүүлүктү берүү үчүн Turbo Pascal тилинде атайын кошуп алуу оператору – with (5-бапты кара) бар. Ушул операторду колдонуу менен жогоруда келтирилген операторлор төмөнкүдөй көрүнүштө жазылат.

```
with  Group_KB51 [1]   do
   begin
     Name         := ‘Nazgul’;
     Year         := 1990;
     Marks. Prog     := 5;
     Marks             := Group_KB52[1] .Marks;
   end;
   Group_KB51[5] :=    Group_KB52[7];
   Group_KB51    :=   Group_KB52;
```

“Жазуу” тибиндеги берилгендердин структурасын информацилар менен толтурууда мына муну эсте тутуу зарыл:

Тексттик файлдардан (клавиатурага дагы Input текстик файлы тиешелеш коюлат) кээ бир стандарттык типтеги гана берилгендерди (баптын башталышын кара) кийрүүгө болот. Ошондуктан Read жана Readln операторлорунда файлдан (из файла) тиби боюнча кийрүүгө уруксат берилген эң ички талаалардын гана идентификаторлору жайгаша алат.

Варианттык жазуулар

Биз жогоруда караган TStudentCard тиби бир семестр үчүн төрт гана бааны кармап турат. Бирок кийинки семестрлерде башка дисциплиналардан экзамен коюлушу мүмкүн. Проблема пайда болот: TStudentCard жазуусунда бизди (биздин мисал үчүн) акыркы семестр үчүн гана баалар кызыктырат, бирок ар бир семестрде башка-башка предметтерден экзамен тапшырылат. Эгерде фиксирленеген кадимки жазууларды пайдалансак, анда TStudentCard тибине окутуу

мезгилиндеги баардык дисциплиналар үчүн талааларды кошууга туура келет. Бул болсо TStudentCard тибиндеги ашыкча информациялардын пайда болушуна жана программанын окулушунун начарлашына алып келет. Андан сырткары, эс экономдуу пайдаланылбай калат, анткени ар бир студент үчүн эс, анын бааларынын талаалары үчүн бөлүнүп берилет, ал эми иш жүзүндө (биздин караган учурубузда) акыркы семестр үчүн бааларды баяндоочу бир нече талаалар гана пайдаланылат.

Мындай учурларда варианттык жазууларды пайдалануу максатка ылайыктуу. Анда деле фиксирленген жазуулар сыяктуу мүмкүн болгон бардык учурлар үчүн талаалар баяндалат, бирок талаалардын альтернативдик тайпалары бир кыйла көрсөтмөлүү чектерге бөлүнөт жана эс берилген моментте конкреттүү зарыл болгон вариант үчүн гана ажыратылат.

Эгерде TStudentCard тибин экинчи семестр үчүн баалар менен кеңейтсек, анда ал варианттык жазуу формасында төмөнкүдөй көрүнүшкө ээ болот:

```
type
    TMarksSem1 = record
        MathAnal1          :Byte;  {Мат.анализ}
         SyzAl             :Byte;  {Сыз.алгебра}
         Prog1             :Byte;  {Программалоо}
         Phys              :Byte;  {Физика}
    end;
    TMarksSem2  =  record
            MathAnal2      : Byte; {Мат.анализ}
            Electron       : Byte;  {Электроника}
            Prog2          : Byte;  {Программалоо}
            DigAutom       : Byte;  {Циф.автоматтар теориясы}
    end;
   TVarStudentCard = record
                { Жазуунун фиксирленген бөлүгү }
              SurName     :String20;          {Фамилиясы}
              Name              : String20;          {Аты       }
               FatherName       : String20;          {Атасынын аты}
               Year             : Integer;           {Туулган жылы}
                HomeAddress  :String;             {Үй дареги}
         GroupCode         :String7;            {Группанын шифри}
                { Жазуунун варианттык бөлүгү }
     Case Semestr    :Byte    of
       1: (MarksSem1: TmarksSem1); {Биринчи семестр үчүн баалар}
       2: ( MarksSem2 : TmarksSem2) ; {Экинчи семестр үчүн баалар}
   end;
```

Варианттык жазуулар эки бөлүктү камтып турат: биринчи бөлүк – кадимки фиксирленген жазуу; экинчи бөлүк – белги талаасынан жана бир же бир нече варианттык компоненттерден турган варианттык жазуу. Варианттык компоненттер конкреттүү "жазуу" тибиндеги элементке белги талаасынын маанисинен көз каранды түрдө альтернативдик принцип боюнча киргизилет.

TVarStudentCard тибинде Semestr талаасы – белги талаасы, ал эми MarksSem1 жана MarksSem2 талаалары – берилген жазуунун альтернативдик варианттары болот.

Эгерде Group_KB51 жана Group_KB52 өзгөрүлмөлөрү жогорууда көрсөтүлгөндөй жарыяланса, анда 3 деген иреттик номердеги бир студенттин карточкасынын экземплярын биринчи семестр үчүн информациялар менен толтуруу, мисалы төмөнкүдөй көрүнүшкө ээ болушу мөмкөн:

```
with Group-KB 52  [3]  do
 begin
      { Жазуунун фиксирленген бөлүгүн толтуруу }
           SurName       :=   'Осмоналиева';
           Name           :=   ' Назгүл';
             FatherName  :=  ' Абдикамиловна;
           Year            :=   1990;
     HomeAddress:= 'Ош ш, Зайнабетдинов көчөсү, 20-үй, 7-кв';
     GroupGode := 'КВ-51';
            { Жазуунун варианттык бөлүгүн толтуруу }
    Semestr1      :=  1;       { Биринчи семестрдин белгисин коюу}
    with MarksSem1    do   {Биринчи семестр үчүн баалар          }
    begin
       MathAnal1   :=5;
       SyzAL          :=5;
       Prog           :=4;
       Phys           :=4;
     end;
  end;
   . . .
```

Акырында варианттык жазуулардын колдонуунун бир ыңгайсыз жагын белгилеп кетебиз. Мындай жазуулардын варианттык компоненттеринде бирдей идентификаторлорду пайдаланууга болбойт.

Мисалы, төмөнкү мисалда «Мат. анализ» предмети үчүн эки альтернативдик вариант тең бирдей MathAnal идентификаторун,

«Программалоо» предмети үчүн бирдей Prog идентификаторун кармап турат, бул болсо туура эмес.

Катасы бар **мисал.**

```
TStudentCard = record
      {Жазуунун фиксирленген бөлүгү}
      . . .
      {Жазуунун варианттык бөлүгү}
    case  Semestr  :  Byte  of
     1:    (MathAnal  :  Byte; {Мат.анализ}
                  SyzAl  :  Byte;   {Сыз.алгебра}
                  Prog   : Byte;   {Программалоо}
                  Phys   : Byte);  {Физика}
    2:    (MathAnal : Byte;    {Мат. анализ}
           Electron     : Byte;   {Электроника}
           Prog         : Byte;   {Программалоо}
           DigAutom  : Byte );   {Цифр.автомат.теориясы}
      end;
```

9.4. Типтердин биргелешүүчүлүгү

Типтердин биргелешүүчүлүгү эки түрдүү болот:

- туюнтмалардагы биргелешүүчүлүк;
- ыйгаруу боюнча биргелешүүчүлүк.

9.4.1. Туюнтмалардагы биргелешүүчүлүк

Туюнтмалардын ичиндеги амалдарды аткарууда ар түрдүү типтердеги операнддар учурашы мүмкүн. Ар бир амал өзүнүн уруксат берилген ар түрдүү типтеги операнддардын комбинацияларынын жыйындысына ээ.

Амалдар үчүн уруксат берилген операндардын типтери 4 - бапта каралган.

Ар түрдүү типтеги операнддардын уруксат берилген (допустимый) комбинациялары төмөнкү эрежелер боюнча аныкталат:

- Эки операнддын тең типтери бирдей болот;
- Эки операнддын тең типтери же чыныгы, же бүтүн сандык тип;
- Бир операнддын тиби экинчи операнддын тибинин камтылуучу диапазону болот;
- Эки операнддын тең типтери бир эле негизги типтин кесиндилери болот.
- Эки операнддын тең типтери базалык типтер менен биргелешкен көптүктөр болот;

- Бир операндын тиби жолчолук, ал эми экинчисиники жолчолук же Pchar тибинде болот;
- Бир операндын тиби – Pointer тиби, а экинчисиники – каалагандай пайдалануучулук көрсөткүчтүк тип;
- Бир операндын тиби – Pchar тиби, а экинчисиники – нөль базалуу символдук массив ({$X+} дедективасы болгон учурда гана);
- Эки операндын тең типтери толук окшош көрсөткүчтөк тип болот ({$X+} директивасы болгон учур гана);
- Эки операндын тең типтери, бирдей сандагы жана тиешелештиктеги параметрлерге ээ болгон процедураларды (функцияларды), (функциялар үчүн дагы жыйынтыктардын окшош типтерине ээ болгон) баяндоочу процедуралык типтер болот.

9.4.2. Ыйгаруу боюнча биргелешүүчүлүк

Ыйгаруу боюнча биргелешүүчүлүк туюнтмалардагы биргелешүүчүлүккө караганда эрежелерди сактоону анча талап кылбайт жана типтердин автоматтык түрдө өзгөртүп түзүлүшүнүн негизги спектрине жол коет. Компилятор ыйгаруу операторлорун иштеп чыгып жатканда жана параметрлердин маанилерин берүүдө ыйгаруу боюнча биргелешүүчүлүк эрежелерин колдонот.

Т2 тибиндеги туюнтманын жыйынтыгынын мааниси, ыйгаруу операторунун сол жагында турган Т1 тибинде А өзгөрүлмөсү менен ыйгаруу боюнча биргелешүүчү болот, башкача айтканда төмөнкү баяндоолор болгон учурда

```
Var  A: T1;
     B: T2;
```

төмөнкү оператор уруксат берилген (допустимый) болот

```
A: = B+B;
```

эгер төмөнкү шарттардын бирөөсү аткарылса:

- Т1 жана Т2 типтери, файылдык же структуралык болбогон, файылдык тип менен байланышпаган (не опирающится) теңдеш типтер болушса;
- Т1 жана Т2 биргелешкен ираттик типтер болушса, жана Т2 тибинин маанилеринин диапозону Т1 тибинин мүмкүн болгон маанилеринин камтылуучу диапозону болсо;

- T1 жана T2 – чыныгы типтер болушса, жана T2 тибинин маанилеринин диапозону T1 тибинин мүмкүн болгон маанилеринин камтылуучу диапозону болсо;
- T1 – чыныгы тип, ал эми T2 - бүтүн сандык тип болсо;
- T1 жана T2 – жолчолук типтер болушса;
- T1 – жолчолук тип, ал эми T2 - символдук тип (char) болсо;
- T1 жана T2 - биргелешкен көптүктүн (множественный) типтер болушса, жана T2 тибинин маанисинин бардык мүчөлүрү T1 тибинин мүмкүн болгон маанилеринин диапозонуна туура келсе;
- T1 жана T2 - биргелешкен көптүктүн (множественный) типтер болушса;
- T1 - Pchar тиби, ал эми T2 - жолчолук константа ({$X+} директивасы болгон учурда гана) болсо;
- T1 - Pchar тиби, ал эми T2 - нөль базалуу символдук массив ({$X+} директивасы болгон учурда гана) болсо;
- T1 жана T2 - биргелешкен процедуралык типтер болушса;
- T1 - процедуралык тип болуп,ал эми T2 - жыйынтыктын тиби, параметрлердин саны жана параметрлердин типтеринин арасындагы тиелештиги боюнча толук окшош болгон процедура же функция болсо;
- T2 объектик тиби T1 объекттик тиби менен ыйгаруу боюнча биргелешкен болот, эгерде T2 тиби T1 тибинин насилине (потомок) таандык болсо;
- T2 объектин көрсөтүп турган P2 көрсөткүчүнүн тиби, T1 объектин көрсөтүп турган P1 көрсөткүчүнүн тиби менен ыйгаруу боюнча биргелешкен болот, эгерде T2 тиби T1 тибинин насили (потомок) болсо.

Контролдук тапшырмалар

1. Берилген чыныгы ***a, b (a>b)*** жана бүтүн ***n*** (***n*** **>0**) үчүн үч $\mathbf{f_1(x_i)}$, $\mathbf{f_2(x_i)}$, $\mathbf{f_3(x_i)}$ функцияларынын маанилерин аргументтин берилген маанилери үчүн эсептегиле

$$x_i = a + i \cdot h ,$$

мында $h = (b - a)/n , i = 0,1 , 2 , \dots n$

$a = 0, b = 2\pi , n = 30.$

$$f_1 = e^{-x} + Sin(x)$$

$$f_2 = \sqrt{x^3} + \frac{1}{x+2}$$

$$f_3 = \frac{x^2 + 3x - 5}{4x + 2}$$

Жыйынтыкты печатка бардык чоңдуктар үчүн маанилерди өз-өзүнчө графаларга жайгаштыруу менен таблица көрүнүшүндө чыгаргыла.

2. Бүтүн сандардын вектору (бир өлчөмдүү массиви) берилген. Вектордун нөлдүк элементтери менен катар жайлашкан эң узун чынжырчадагы элементтердин санын ошондой эле бул чынжырчадагы биринчи жана акыркы элементтердин индекстерин аныктагыла.

3. Квадраттык матрица берилген. Ушул матрицанын диагоналдык элементтерин

а) сыйлыгыштыруу

б) тандоо

в) алмашуу

усулдарын пайдалануу менен кемүү тартибинде иреттегиле.

4. Чыныгы сандардын тик бурчтуу матрицасы (эки өлчөмдүү массиви) берилген. Анын мамычаларынын максималдык элементеринен түзүлгөн векторду печатка чыгаргыла.

5. Узундугу **n** болгон символдордун чынжырчасы берилген.

(‘0’ > ‘1’ , ‘1’> ‘2’ , ‘2’> ‘3’…)

Андагы цифра болуп эсептелген ар бир символду мааниси боюнча андан кийин келген цифра болуп эсептелген символго алмаштыргыла.

6. Бүтүн сандардын тик бурчтуу матрицасы берилген. Экилик издөө усулу менен матрицанын ар бир жолчосунда берилген X элементинин бар экендин ар бир жолчо үчүн өз-өзүнчө аныктагыла.

7. Символдук типтеги беш A, B, C, D, E көптүктөрү берилген. Төмөнкүчө аныкталган

$X = (A \cup B) \cap (C \cup D) \cap E$, эгер $(A \cap B) \subset C$ шарты чын болсо

жана андай болбогон учурда $X = A \cup (B \cap C) \cup (D \cap E)$

X көптүгүнүн элементтерин тапкыла.

8. Группадагы студенттердин акыркы үч семестр үчүн бааларын өз ичине алып турган жазуулар массивин баяндагыла. Үч сессиянын баарын тең жалаң «эң жакшы» деген бааларга тапшырган студенттердин фамилияларын аныктагыла.

10. ПРОЦЕДУРАЛАР ЖАНА ФУНКЦИЯЛАР

Билим алуу – ийне менен кудук казгандай.
(Эл макал-лакаптары)

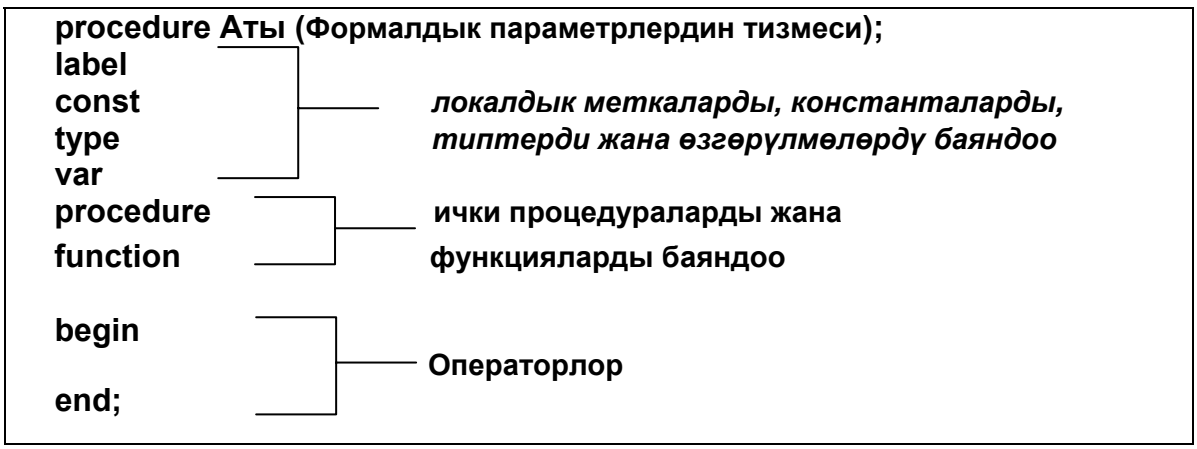
10.1. Процедуралардын жана функциялардын структурасы

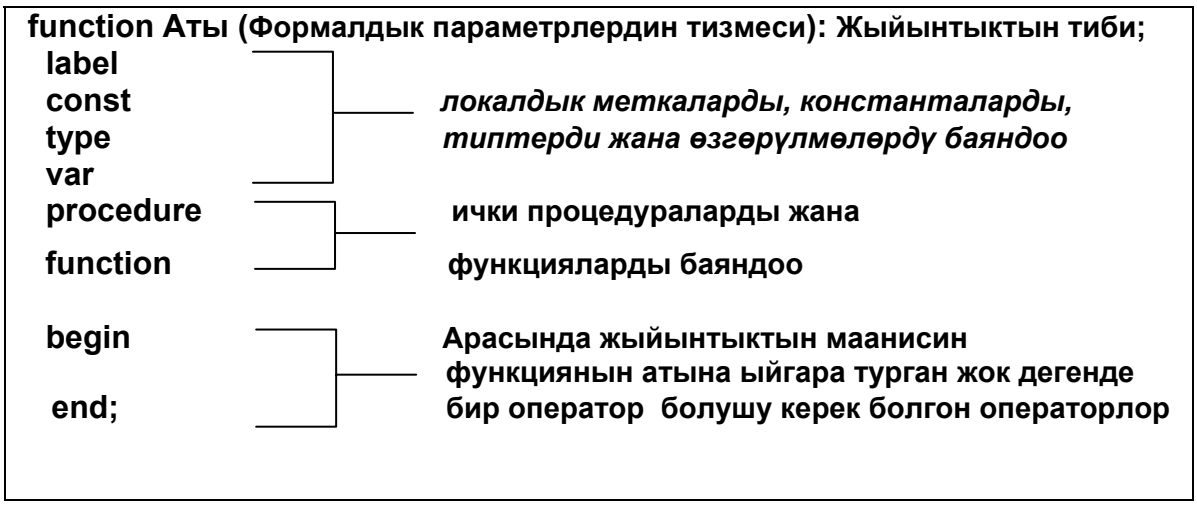
Адатта татаал маселелерди чечүүгө арналган программаларды түзүүдө ал маселени андан кичинерээк маселелерге, андан ары зарыл болсо пайда болгон кичинерээк маселелерди дагы да кичинерээк маселелерге жана ушул сыяктуу жеңил программалануучу элементардык маселелерге чейин декомпозициялоо аткарылат.

Программаларды ушундай кылып майда бөлүктөргө ажыратуу үчүн Turbo Pascal тилинде ар түрдүү каражаттар бар. Чоң маселелерди майда маселелерге ажыратуунун жогорку деңгээлинде – модулдар, төмөнкү деңгээлинде - көпчүлүк учурда процедуралар жана функциялар турат. Объекттик ориентирленген усулият программаларды талдоонун ушул эки деңгээлинин экөөнү тең өз ичине алат.

Программалоо тилдеринин көпчүлүгүндө процедуралар жана функциялар негизги, башкы каражат болуп эсептелет. Алардын жардамында кандайдыр бир бирдиктүү аракетти аткаруу үчүн операторлордун тайпасын компоновкалоого болот. Процедураны/функцияны программанын каалаган жеринде туруп чакырууга мүмкүн болуп, ал эсептелинип чыккан жыйынтыкты кайтарып бере алат, жана ага эсептөөлөрдү аткаруу үчүн керектүү болгон информацияны берүүгө болот. Процедура/функция иштей башташы үчүн аны чакыруу (активдештирүү) керек.

Процедуралар жана функциялар операторлордон, локалдык берилгендерден жана ички процедуралардан жана функциялардан турат. Процедураны жана функцияны баяндоонун структурасы төмөндөгүдөй көрүнүшкө ээ:





Бул сүрөттөрдөн процедура менен функцияны баяндоодогу айырмачылык - бөрктө жана операторлор бөлүгүндө гана экендиги көрүнүп турат. Ушул айрымачылыктарды демострациялоо максатында бир эле маселенин эки түрдүү чечилишин, бирөөндө процедуранын, экинчисинде - функциянын жардамында чечилишин келтиребиз.

Маселе. **n** ченемдүү, элементтери чыныгы сандар болгон бир өлчөмдүү массив берилген. Вектордун терс эмес элементтеринин суммасын табуу керек. Процедураны колдонуу менен жазылган программа төмөнкү көрүнүштө болот.

```
program PSumma;
const  n =30;    {Массивдин узундугу}
type   TVector = array [l..n] of real;
Var    Vector: TVector;
       i: Integer;
procedure Summa (Vec: TVector; Len: Integer; Name: string);
Var
    i: Integer;
   S: real;
begin
    S:=0;
    for i:= 1 to Len do
If Vec[i]>=0 then S:=S+Vec[ i ];
Writeln ('Терс эмес элементтеринин суммасы ', Name, ' = ', S:7:2)
end;
begin
    Writeln ('Вектордук элементтерин киргизгиле:');
    for i:= 1 to n do read (Vector [i]); Readln;
    { Процедураны чакыруу өзүнчө оператор аркылуу аткарылат }
    Summa (Vector, n, 'Vector');
end.
```

Ушул эле маслени чечүүчү программаны функцияны колдонуу менен төмөнкүчө жазууга болот.

```
program FSumma;
const
        n = 30; {массивдин узундугу}
type
       TVector = array [l..n] of real;
var
        Vector   : TVector;
         Sum     : Real;
          i      : Integer;
function Summa (Vec: TVector; Len: Integer): Real;
var
    i    : Integer;
   S    : Real;
begin
     S:=0;
     for i:= 1 to Len do
if Vec[i] >= 0 then S:=S+Vec[i];
Summa:=S   {функциянын атына жыйынтыктын маанисин    }
           {ыйгаруу оператору функциянын телосундагы  }
           {сөзсүз көрсөтүлүүчү оператор болуп саналат  }
end;
begin
      Writeln ('Массивдин элементтерин киргизгиле');
      for i:= 1 to n do read ( Vector [i] );  Readln;

{функцияны чакыруу, туюнтманы коюга уруксат}
{берилген жерде аткарылышы мүмкүн, айрым учур катары}
{ыйгаруу опреаторунун оң  жагында}

Sum:= Summa ( Vector, n );
Writeln ('Vector дун терс эмес элементтеринин  суммасы=',
                                                Sum:7:2);
{же түздөн - түз Writeln процедурасынын чыгаруу элементи катары}
Writeln ( 'Vector векторунун терс эмес элементтеринин
                              суммасы=', Summa ( Vector, n )  )
end.
```

10.2. Процедураларды жана функцияларды колдонууда идентификаторлордун аракет этүү аймагы (көрүнүктүүлүк аймагы)

Идентификатордун ***аракет этүү аймагы*** деп ушул идентификатор пайдаланылышы мүмкүн болгон программанын аймагы аталат.

Идентификаторлордун аракет этүү области алар баяндалган жер менен аныкталат. Эгер идентификаторлорду бир гана процедуранын же функциянын алкагында колдонууга уруксат берилсе, анда мындай идентификаторлор ***локалдык*** деп аталат, ал эми эгер идентификаторлордун аракет этүүсү бир нече (бирден кем эмес) камтылма процедураларга же/жана функцияларга жайылтылса анда мындай идентификаторлор ***глобалдык*** деп аталат.

«Глобалдык» жана «локалдык» түшүнүктөрүн конкреттүү процедурага же функцияга салыштырмалуу түшүнүү керек

```
program Sc_p;
 var A0, B0, C0: Integer;
   procedure P1;
    var A1, B1, C1:Integer;
      procedure P2;
       var A2, B2, C2:Integer;
       begin
          Глобалдык A0, B0, C0, A1, B1, C1
          ошондой эле локалдык A2, B2, C2
       идентификаторлорун колдонууга уруксат
       end;
    Begin
     Глобалдык A0, B0, C0 жана
      локалдык  A1, B1, C1 идснтификаторлорун
      колдонууга уруксат
   end;
Begin
 A0, B0, C0 идентификаторлорун
 гана   колдонууга уруксат
end.
```

Бул келтирилген мисалда A0, B0, C0 идентификаторлору программада пайдаланылуучу бардык процедуралар жана функциялар үчүн глобалдык болушат.

A1, B1, C1 идентификаторлору P1 процедурасынын ичинде баяндалган бардык процедуралар жана функциялар үчүн (биздин мисалда P2 процедурасы үчүн) глобалдык болушат.

Эң ички P2 процедурасында жарыяланган A2, B2, C2 берилгендери локалдык гана болушат.

Процедуралардын жана функциялардын идентификаторлору үчүн аракет этүү областын аныктоо эрежесин келтиребиз:

- Процедуранын/функциянын ичинде аныкталган бардык идентификаторлор аракет этет;
- Эгерде аттары процедуранын/функциянын ичинде жарыяланган аттардан айрымаланса, анда курчап турган контексттин бардык идентификаторлору аракет этет;
- Процедуранын/функциянын локалдык идентификаторлору сырткы курчоодо эч качан аракет этпейт.
- Глобалдык жана локалдык идентификаторлордун аттары дал келип калган учурда ички локалдык идентификатор гана аракет этет.

Биринчи үч эрежени жогорку келтирилген мисал түшүндүрө алат, ал эми төртүнчү эрежени түшүнүү үчүн дагы бир мисал келтиребиз.

```
Program Eq_N;
Var A, B, C: integer;
 Procedure P1;
  Var A, B, C: Real;
   Procedure P2;
    Var A, B, C: Char;
    Begin
    A, B, C лар символдук өзгөрүлмөлөр
    катары колдонулат

   End;
    Begin
    A, B, C лар чыныгы өзгөрүлмөлөр
    катары колдонулат

    end;
Begin
A, B, C лар бүтүн өзгөрүлмөлөр
катары колдонулат
 End.
```

Башка сөз менен айтканда, сырткы процедуралардын берилгендери менен аттары боюнча дал келген идентификаторлор менен берилгендерди ички процедураларда баяндоо сырткы

идентификатордун аракетин жокко чыгарат жана типтери боюнча дал келеби, келбейби андан кез карандысыз түрдө өзүнүн локалдык баяндоолорун киргизет.

Локалдык берилгендер процедураны/функияны чакырган учурда түзүлөт жана ал аткарылып жаткан учурда гана жашайт. Локалдык берилгендер үчүн эсти бөлүү автоматтык түрдө процедуранын/функциянын аткарылышынын башында болуп өтөт, ал эми бул эсти бошотуу – процедура/функция аткарылып бүтөрү менен болот.

Процедуранын/функциянын телосунда жайгашкан операторлор ал процедуранын/функциянын локалдык берилгендерине (константаларына жана өзгөрүлмөлөрүнө) кайрыла жана алардын маанилерин өзгөртө алат.

Бирок эске тутуучу нерсе, локалдык берилгендердин маанилери процедура/функция иштеп турган учурда гана жашайт. Ал аяктары менен процедуранын/функциянын операторлору тарабынан локалдык берилгендердин маанилеринин бардык өзгөрүүлөрү эсти бошотуу менен бирге жок болот.

10.3. Параметрлерди берүү жолдорун классификациялоо

Процедураны/функцияны активдештирүүдө ага параметрлерди берүү мүмкүн, бирок мындай берүүдө процедуранын/функциянын бөркүндө баяндалган бардык параметрлердин чакыруу операторунда көрсөтүлүшүнө көңүл буру зарыл.

Процедураны/функцияны баяндоодо анын бөркүндө көрсөтүлүүчү параметрлер ***формалдык параметрлер*** деп аталат, ал эми процедураны/функцияны чакырган учурда көрсөтүлүүчү параметрлер ***иш жүзүндөгү*** (***фактические***) параметрлер деп аталат.

Параметрлерди берүүнүн корректтүүлүгү алардын процедуранын бөркүнө саналып жазылыш тартибине жана тиешелеш иш жүзүндөгү жана формалдык параметрлердин ортосундагы ыйгаруу боюнча биргелешкендикке негизделген. Параметрлердин аттарынын аракет этүү сферасы локалдык берилгендердики сыяктуу эле.

Көпчүлүк процедуралар бир нече параметрлерге ээ болушат. Программистке коюлган маселе – ал чакырууда көрсөтүп жаткан параметрлер (иш жүзүндөгү параметрлер) мааниси боюнча формалдык параметрлерге тиешелеш келишине ишенүү. Компоненттер болсо жеңил көрүнүп турган учурларды гана -

параметрлердин санынын туура эместигин же типтердин биргелешпегендигин текшере алат.

Программалоо тилдеринде реализацияланышы мүмкүн болгон параметрлерди берүү жолдорунун классификациясын карайбыз.

Көңүл бурчу нерсе, бул TURBO PASCAL тилиндеги параметрлердин эмес, а жалпы классификация. Классификация үчүн киргизилген value, addr, in, out, inout белгилөөлөрү Turbo Pascal тилинде пайдаланылбайт.

Параметрлерди төмөнкүлөр боюнча айрымалап кароого болот:
1) Берүү механизми боюнча:

а) мааниси боюнча берүү (value); б) адреси боюнча берүү (шилтеме боюнча) (addr)

2) Чакыруучу жана чакырылуучу процедуралардын/функциялардын өз ара аракет этүүсү боюнча:

а) кирүү параметри катары гана (in) б) чыгуу параметри катары гана (out) в) кирүүчү да чыгуучу да параметр катары (inout)

Ушул айырмачылыктардан улам назарияттык жактан алып караганда параметрлерди берүүнүн 6 жолу болушу мүмкүн:

1. value in;
2. value out;
3. value inout;
4. addr in;
5. addr out;
6. addr inout;

Параметрлерди берүүнүн ушундай 6 түрдүү жолу бар болушу мүмкүн экендигине карабастан программалоо тилдеринде, эреже катары, эки-үч түрдүү гана жолу пайдаланылат. Turbo Pascal да параметрлерди берүүнүн жолдорун ичинен биринчиси (value in), төртүнчүсү (addr in) жана алтынчысы (addrinout) реализацияланган.

Ушул параметрлерди берүүнүн 6 жолунун ар биринин кыскача манызын карап көрөбүз.

10.3.1. Value in параметрлери (TP тилинде реализацияланган)

1. *Процедураны/функцияны чакырууда:*

а) формалдык параметрлер жана локалдык берилгендер үчүн эсте орун ажыратуу (локадык берилгендер үчүн стекте же эстин атайын областында) аткарылат.

б) иш жүзүндөгү параметрлердин маанилерин, формалдык параматрлер үчүн ажыратылып коюлган эстин бөлүгүнө көчүрүү аткарылат.

2. *Процедура/функция иштеп жаткан учурда:*

а) формалдык параметрлердин маанилеринин өзгөрүшү, иш жүзүндөгү параметрлер үчүн ажыратылган эстин ячейкаларынын мазмунуна эч кандай таасирин тийгизбейт.

3. *Процедура/функция аяктаганда*:

а) формалдык параметрлер жана локалдык берилгендер үчүн ажыратылган эс тазаланат.

б) процедуранын иштөө процессинде алынган формалдык параметрлердин жаңы маанилери эстин тазаланышы менен бирге жок болот.

10.3.2. Value out параметрлери

1. *Процедураны/функцияны чакырганда:*

а) формалдык параметрлердин жана локалдык берилгендердин маанилери үчүн, ошондой эле иш жүзүндөгү параметрлердин адрестерин сактоо үчүн эсте орун бөлүштүрүү (локалдык берилгендер үчүн стекте же эстин атайын аймагында) аткарылат.

б) иш жүзүндөгү параметрлердин адрестерин (бирок маанилерин эмес!) алар үчүн бөлүнгөн эске көчүрүү аткарылат.

в) формалдык параметрлердин маанилери түздөн түз чакыруудан кийин аныкталбаган.

г) иш жүзүндөгү параметрлер катары константаларды пайдалануга тыйуу салынат.

2. *Процедура/функция иштеп жаткан учурда:*

а) формалдык параметрлердин маанилерин кандайдыр бир маанини алгачкы ирет ыйгаруу аткарылмайынча пайдалануга болбойт.

б) формалдык параметрлердин маанилеринин өзгөрүшү, бир убакта тиешелүү иш жүзүндөгү параметрлердин маанилеринин өзгөрүшүнө алып келбейт.

3. *Процедура/функция аяктаганда:*

а) адегенде иш жүзүндөгү параметрлердин көчүрүлүп алынган адрестерин пайдалануу менен формалдык параметрлердин жыйынтык маанилерин иш жүзүндөгү параметрлердин эстеги ячейкаларына жазуу аткарылат.

б) процедуранын/функциянын иши үчүн ажыратылган эс тазаланат.

10.3.3. Value inout параметрлери

1. *Процедураны/функцияны чакырганда:*

а) формалдык параметрлердин жана локалдык берилгендердин маанилери үчүн, ошондой эле иш жүзүндөгү параметрлердин адрестерин сактоо үчүн эсте орун бөлүштүрүү (локалдык берилгендер үчүн стекте же эстин атайын аймагында) аткарылат.

б) иш жүзүндөгү параметрлердин адрестерин жана алардын маанилерин да алар үчүн бөлүнгөн эске көчүрүү аткарылат.

в) иш жүзүндөгү параметрлер катары константаларды пайдаланууга тыйуу салынат.

2. *Процедура/функция иштеп жаткан учурда:*

а) формалдык параметрлердин маанилерин пайдаланууга жана аларды өзгөртүүгө уруксат берилет.

б) формалдык параметрлердин маанилеринин өзгөрүшү бир убакта тиешелүү иш жүзүндөгү параметрлердин маанилеринин өзгөрүшүнө алып келбейт.

3. *Процедура/функция аяктаганда:*

а) Value out парметрине окшош, адегенде иш жүзүндөгү параметрлердин көчүрүлүп алынган адрестерин пайдалануу менен формалдык параметрлердин жыйынтык маанилерин иш жүзүндөгү параметрлердин эстеги ячейкаларына жазуу аткарылат.

б) процедуранын/функциянын иши үчүн ажыратылган эс тазаланат.

10.3.4 Add in параметрлери (TP тилинде раелизацияланган)

1. *Процедураны/функцияны чакырууда:*

а) локалдык берилгендер үчүн жана иш жүзүндөгү параметрлердин адрестерин сактоо үчүн гана (локалдык берилгендер үчүн стекте же эстин атайын областында) эсте орун ажыратуу аткарылат;

б) иш жүзүндөгү параметрлердин адрестерин (бирок маанилерин эмес!) алар үчүн ажыратылып коюлган эске көчүрүү аткарылат;

2. *Процедура/функция иштеп жатканда:*

а) формалдык параметрлердин маанилерин өзгөртүүгө тыюу салынат;

б) формалдык параметрлердин маанилерин баштапкы берилгендер катары гана пайдаланууга уруксат берилген.

в) формалдык параметрлердин маанилери, көчүрүлүп алынган адрестерди пайдалануу менен, тикеден-тике иш жүзүндөгү параметрлер үчүн ажыратылган эстен тандалып алынат.

3. *Процедура/функция аяктганда:*

а) чакырылуучу процедуранын/функциянын ушундай параметрлер аркылуу чакырылуучу процедура/функцияга таасири жок.

б) процедуранын/функциянын иши үчүн ажыратылган эс тазаланат.

10.3.5. Addr out параметрлери

1. *Процедураны/функцияны чакырууда:*

а) локалдык берилгендер жана иш жүзүндөгү параметрлердин адрестерин сактоо үчүн гана эсти (локалдык берилгендер үчүн стекте же эстин атайын аймагында) ажыратуу аткарылат;

б) иш жүзүндөгү параметрлердин адрестерин (бирок маанилерин эмес!) алар үчүн ажыратылган эске көчүрүү аткарылат;

в) формалдык парметрлердин маанилери түздөн-түз чакыруудан кийин аныкталбаган;

г) константаларды иш жүзүндөгү параметрлер катары пайдалануга тыюу салынат;

2. *Процедура/функция иштеп жатканда:*

а) кандайдыр бир маанинин алгачкы ыйгарылышы аткарылмайынча формалдык параметрлердин маанилерин пайдалануга тыйу салынат;

б) формалдык параметрлердин маанилеринин өзгөрүшү, көчүрүлүп алынган адрестерди пайдалануу менен, тиешелүү иш жүзүндөгү параметрлердин эстеги ячейкаларында аткарылат.

3. *Процедура/функция аяктаганда:*

а) жыйынтыктарды атайын көчүрүү талап кылынбайт, анткени жыйынтыктар дароо эле иш жүзүндөгү параметрлердин эстеги ячейкаларында алынат;

б) процедуранын/функциянын иши үчүн ажыратылган эс тазаланат.

10.3.6. Addr inout параметлери

1. *Процедураны/функцияны чакырууда:*

а) локалдык берилгендер үчүн жана иш жүзүндөгү параметрлердин адрестерин сактоо үчүн гана (локалдык берилгендер үчүн стекте же эстин атайын областында) эсти ажыратуу аткарылат;

б) иш жүзүндөгү параметрлердин адрестерин (бирок маанилерин эмес!) алар үчүн ажыратылган эске көчүрүү аткарылат;

в) константаларды иш жүзүндөгү параметрлер катары пайдалануга тыюу салынат;

2. *Процедура/функция иштеп жатканда:*

а) мындай көрүнүштөгү параметрлерди пайдаланууга эч кандай чектөөлөр коюлбайт;

б) формалдык параметрлердин маанилеринин өзгөрүшү, көчүрүлүп алынган адрестерди пайдалануу менен тиешелүү иш жүзүндөгү параметрлердин эстеги ячейкаларында аткарылат.

3. *Процедура/функция аяктаганда:*

а) жыйынтыктарды атайын көчүрүү талап кылынбайт, анткени формалдык параметрлер менен болгон бардык аракеттер иш жүзүндөгү параметрлердин эсинин ячейкаларынын үстүнөн түздөн-түз аткарылды.

б) процедуранын/функциянын иши үчүн ажыратылган эс тазаланат.

10.4. Turbo Pascal тилинде параметрлерди берүү

Жогоруда каралган параметрлерди берүүнүн алты ыкмасынын ичинен Turbo Pascal тилинде үчөө ишке ашырылган.

1. *Value in көрүнүшүндөгү параметрлер.*

Turbo Pascal тилинде мындай параметрлер ***параметр-маанилер*** деп аталат. Процедуралардын/функциялардын бөрктөрүн баяндоодо параметр-маанилердин индетификаторлорунун алдынан **кошумча кызматчы сөздөр коюлбайт.**

2. *Addr inout көрүнүшүндөгү параметрлер*

Turbo Pascal тилинде мындай параметрлер ***параметр-өзгөрүлмөлөр*** деп аталат. Процедуралардын/функциялардын бөрктөрүн баяндоодо параметр-өзгөрүлмөлөрдүн индетификатор-лорунун алдынан Var **кызматчы сөзү коюлат.**

3. *Addr in көрүнүшүндөгү параметрлер.*

Turbo Pascal тилинде мындай параметрлер ***параметр-константалар*** деп аталат. Процедуралардын/функциялардын бөрктөрүн баяндоодо параметр-константалардын индетификаторлорунун алдынан Const **кызматчы сөзү коюлат.**

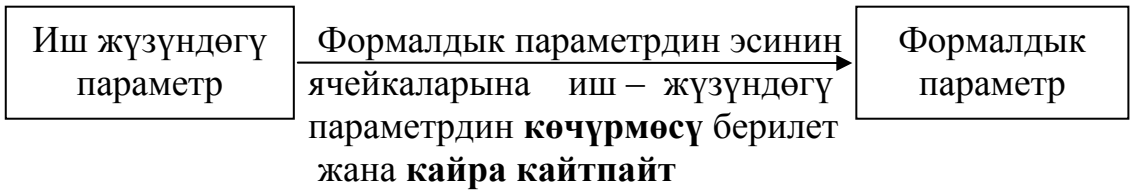
Биринчи эки көрүнүштөгү параметрлер (параметр-маанилер жана параметр-канстаталар) Turbo Pascal тилинин бардык реализациялары жана версиялары үчүн стандарттык болуп эсептелет. Үчүнчү түрдөгү (параметр-константалар) параметрлер тилдин Turbo Pascal 7.0 версиясына киргизилген жаңылык.

10.4.1. Параметр-маанилер

Параметр-маанилердин баяндоосун кармаган процедуранын бөлүгү төмөнкү көрүнүштө болот:

```
Procedure MyProc(Par1, Par2: Type1;  Par3, Par4: Type2);
```

Параметр-маанилердин иштөө механизмин жөнөкөйлөтүп төмөнкүдөй схема менен көрсөтсөк болот:



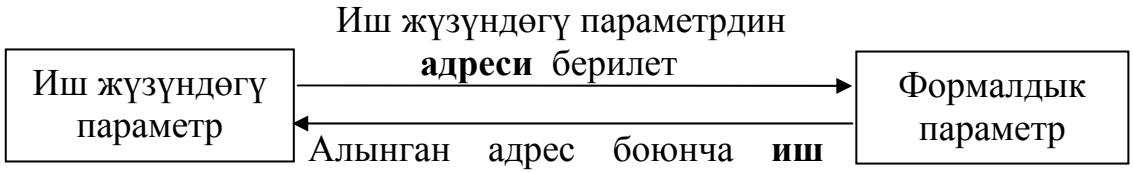
Иш жүзүндөгү парамет-маанилер катары ар турдуу типтеги өзгөрүлмөлөр жана константалар пайдаланылышы мүмкүн. Файлдык типтер жана файлдык типтерге негизделген типтерди гана пайдаланууга уруксат жок.

10.4.2. Параметр-өзгөрүлмөлөр

Параметр-өзгөрүлмөлөрдүн баяндоосун кармап турган процедуранын бөркү төмөнкүдөй көрүнүштө болот:

```
Procedure MyProc(var Par1, Par2: Type1; var Par3, Par4: Type2);
```

Параметр-өзгөрүлмөлердүн иштөө механизмин жөнөкөйлөтүп төмөнкү схема боюнча көрсөтүүгө болот:



б.а. жыйынтыкты кайтаруу болот.

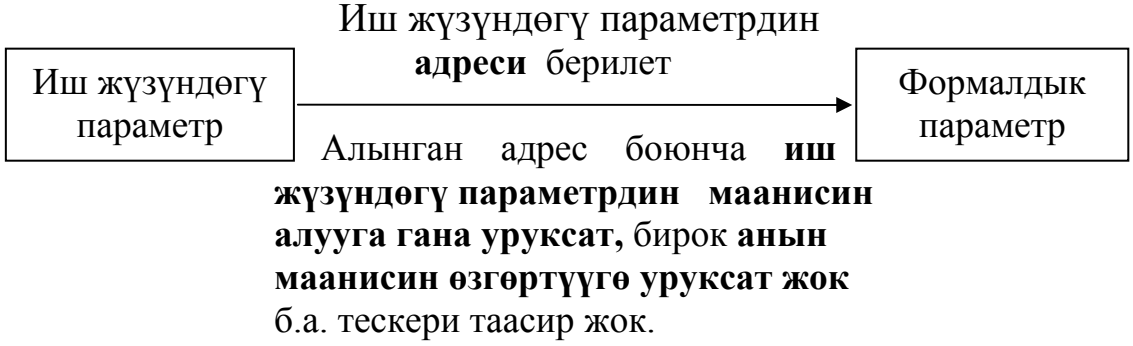
Иш жүзүндөгү параметр-өзгөрүлмөлөр катары файлдык типтеги жана файлдык типтерге негизделген типтер менен бирге каалаган типтеги өзгөрүлмөлөр пайдаланылышы мүмкүн, бирок константаларды пайдаланууга уруксат жок.

10.4.3. Параметр-константалар

Параметр-константалардын баяндоосун өз ичине алган процедуранын бөркү төмөнкүдөй болот:

```
Procedure MyProc(const  Par1, Par2: Type1; const  Par3, Par4: Type2);
```

Параметр-константалардын иштөө механизмин жөнөкөйлөтүп төмөнкү схема боюнча көрсөтүүгө болот:



Иш жүзүндөгү параметр-константалар катары ар турдуу типтеги өзгөрүлмөлөр да, константалар да пайдаланыла алат. Файылдык типтер жана ага байланышкан типтерди гана пайдаланууга болбойт. Андан сырткары формалдык параметр-константаларга ыйгарууну аткарууга тыюу салынат жана формалдык параметр-констаталар башка процедураларга/функцияларга иш жүзүндөгү параметр катары бериле албайт.

Параметр-константаларды чоң өлчөмдөгү эсти ээлей турган берилгендердин структурасын берүү талап кылынып, бирок алгоритмдин көз карашы менен алганда параметрлердин баштапкы маанилерин өзгөртүү мүмкүн эмес болгон учурларда пайдалануу максатка ылайыктуу. Натыйжада оперативдик эс экономдуу пайдаланылат жана ошону менен бир убакта баштапкы берилгендердин бүтүндүүлүгү кепилденет.

10.4.4 Типсиз параметрлер

Типсиз параметрлер ***адрес боюнча гана*** б.а. параметр-өзгөрүлмөлөр же параметр-константалар катары гана бериле алат. Типсиз параметрлердин башкы өзгөчүлүгү – процедуранын бөркүндө параметрдин тибинин көрсөтүлүшү жок экендигинде.

Баяндалган типсиз параметрлерди кармаган процедуранын бөркү, параметрлерди берүүнүн ар түрдүү жолдору үчүн төмөнкүдөй көрүнүштө болот:

```
Procedure MyProc(var Par1, Par2; const Par3, Par4);
```

Бирок, эсте тутчу нерсе, типтин көрсөтүлбөгөндүгүнөн улам типсиз формалдык параметрлерди типтештирилген параметрлердей пайдаланууга болбойт. Пайдалануунун алдында формалдык типсиз параметрди кандайдыр бир типке келтирип алуу талап кылынат.

Типтештирилгенге караганда типсиз параметрлер иштөөдө бир топ ийкемдүү, бирок аларды корректүү (туура) пайдалануу

жоопкерчилиги программистке жүктөлөт, анткени компилятор типтердин биргелешкендигин текшере албайт.

Мисал катары элементтери Byte тибинин камтылуучу диапазону болгон каалагандай типте болгон, узундугу каалагандай болгон бир өлчөмдүү масив үчүн кийрүү, чыгаруу жана сорттоо универсалдык процедуралары пайдаланылган программаны карайбыз.

Мында ар бир массив ага канча орун талап кылынса эсте туура ошончо орунду ээлейт.

Анын үстүнө SortVector процедурасы Char тибиндеги массивдерди сортто үчүн ийгиликтүү пайдаланылышы мүмкүн, анткени ASCII символдорунун коддору Byte тибиндеги маанилердин пределинен чыгып кетпейт. Ошол эле учурда InpVector жана PrintVector процедуралары символдук массивдер үчүн туура келбейт, анткени алар сандык маанилерди кийрүү-чыгаруу үчүн багытталган, ал эми символдорду кийрүү-чыгаруу башкача аткарылат.

```
Progpam Sort1;
   uses crt ;
   const
          l = 10;
          m = 15;
          n = 20;
  type
       MaxVector = array [1..MaxInt] of Byte;
       TVecbyte = array [1..l] of  Byte;
       TVec100 =array [1..m] of  1..100;
       TVecChar = array [1..n] of Char;
  var
       ByteVec: TVecByte;
       Vec100: Tvec100;
       CharVec: TVecChar;
        i          :  Word
  Procedura SortVector ( var Vector; len: Word) ;
  var
     Min:  Byte;
     Imin: Word;
     i, J  : Word;
 begin
     for i: = 1 to Len -1 do
     begin
       Min:= MaxVector(Vector)[i]; {Vector параметрин MaxVector}
                                   { тибине келтирүү                }
        Imin:= i;
        for j:= i+1 to len do
```

```
            if MaxVector(Vector)[ j ] < Min then
            begin
              Min:= MaxVector(Vector)[ j ];
              Imin:= j
            end;
        MaxVector(Vector)[ Imin ] := MaxVector(Vector)[ i ];
        MaxVector(Vector)[i ]:= Min;
     end;
  end;
  Procedura InpVector (Var Vector; Len: Word; T :String );
  Var
      i:  Word;
begin  clrscr;
    Write ln(  T,': тибиндеги' ,Len:3,' элементтерди киргизгиле');
    for i:= 1 to len do Read (MaxVector(Vector)[i]);
    readIn;
end;
 Procedure PrintVector ( var Vector: Len: Word);
  var
     i:  word;
  begin
        WriteIn ( 'Сорттолуп коюлган массив:');
        for i: =1 to len do Write (MaxVector(Vector)[i ]:8);
        WriteIn;
        WriteIn ('Enter   ди баскыла...');
        ReadIn;
  end;
  begin
      InpVector ( ByteVec, 1, 'Byte');
      SortVector (ByteVec, 1);
      PrintVector(ByteVec, 1);
       InpVector ( Vec100, m, '1..100');
       SortVector( Vec100, m);
       PrintVector(Vec100, m);
  ClrScr;
   WriteIn ('Char тибиндеги', n:3, 'элементерди киргизгиле');
        for i:= 1 to n do Read(CharVec[ i ] );
        ReadIn ;
    SortVector (CharVec, n);
        WriteIn ('Сорттолгон массив:');
        for i: =1 to n do Write (CharVec[i ]:8);
         WriteIn;
         WriteIn ( 'Enter ди баскыла...');
         ReadIn ;
 end.
```

10.4.5. Ачык параметр-массивдер

Процедуранын бөркүндө ачык параметр-массивдер төмөндөгудөй баяндалат:

Procedure **OpenVector** (**Vector**: array of **TypeVector**);

Ачык параметр-массив – бул параметр-маани, параметр-константа же параметр-өзгөрүлмө болушу мүмкүн.

Берилүүчү иш жүзүндөгү параметрлер тиби боюнча баяндалган typeVector тиби менен дал келиши керек, бирок өлчөмү боюнча ар түрдүү боло бериши мүмкүн: type Vector тибиндеги жөнөкөй өзгөрүлмө жана ошондой эле каалаган өлчөмдөгү массив катары да болушу мүмкүн.

Ар түрдүү өлчөмдөгү массивдерди берүүдөгү ийкемдүүлүк бул массивдерди формалдык параметрлер катары көрсөтүүдөгү бир түспөлдүүлүктүн эсебинен алынат. Бардык формалдык ачык параметр-массивдер процедуранын алкагында автоматтык түрдө бөрктө көрсөтүлгөн типтеги нөл базалуу (төмөнкү чеги нөл болгон) массив катары баяндалат:

array[0 .. n-l] of TypeVector;

Мында n – иш жүзүндөгү параметрдеги элементтердин саны.

Башка сөз менен айтканда иш жүзүндөгү параметр-массивдин индексинин өзгөрүшүнүн чыныгы диапазону индекстин 0 дөн n-1 ге чейинки өзгөрүш диапазонуна чагылат.

Берилген иш жүзүндөгү параметрдин мүнөздөмөлөрүн аныктоо үчүн процедуранын телосунда дайыма 0 ду берүүчү Low стандарттык функциясы, формалдык параметр-массивдеги акыркы элементтин индексин берүүчу High стандарттык функциясы жана параметр-массивдин өлчөмүн берүүчү SizeOf функциясы пайдаланылат.

Ачык параметр-массивдердин жардамында жогорудагы мисалдагыдай проблемаларды чечүүгө болот. Бирок ачык парамерт-массивдер типсиз параметрлерге караганда анчалык ийкемдүү эмес, анткени бул учурда бир типтеги массивдер гана иш жүзүндөгү параметрлер катары боло алат.

Эми типсиз параметрлердин пайдаланылышы менен салыштыруу үчүн, жогорудагы мисал үчүн ачык параметр-массивдерди пайдалануу менен түзүлгөн программаны келтиребиз.

```
Program Sort2;
   ues Crt;
   const m=15; n =20;
   type TVec1 = array[1.. m] of  Byte;  TVec2 = array [ 1.. n ] of  Byte;
   var Vec1 : TVec1;  Vec2 : TVec2;
```

```
      procedyre SortVector ( var Vector: array of Byte);
      var    Min : Byte;
             Imin: Word;
             i, j   : Word;
   begin
       for i: = 0 to High (Vector)  - 1 do
      begin
        Min  : = Vector [i];
        Imin : = i;
        for j  : = i+1 to High (Vector) do
             If Vector [j] < Min  then
           begin
             Min  : = Vector [j];
             Imin : = j
           end;
        Vector [Imin]: = Vector [i];
        Vector [i]: = Min;
        end;
    end;

    procedyre InpVector (var Vector : array of Byte);
    var     i: Word;
    begin ClrScr;
    WriteIn('Byte тибиндеги', (High (Vector )+ 1): 3, 'элементтерди
                                                          киргизгиле');
    for i: = 0 to High (Vector) do Read (Vector [i]);
    readIn;
    end;
  procedyre PrintVector ( var Vector : array of byte);
  var    i: Word;
  begin
     WriteIn (" Сорттолгон массив: ' ) ;
     for i: =0 to Hiigh (Vector) do  Write ( Vector [i] : 8 );
    WriteIn;
    WriteIn ( 'Enter ди баскыла ...');
    ReadIn;
  end;
  begin
      InpVector    (Vec1) ;
      SortVector   (Vec1);
      PrintVector  ( Vec1);
      InpVector   (Vec2) ;
      SortVector  (Vec2);
      PrintVector (Vec2);
end.
```

10.5. Процедуралык директивалар

10.5.1. near жана far директивалары

near жана far директивалары компиляторго процедураны чакыруунун кайсы моделине карай - жакынкыбы (near) же алыскыбы (far) - чыгуу кодун генерациялоо талап кылынарын көрсөтөт. Жакынкы модели боюнча болгондо тезирээк, алыскы болгондо жайыраак аткарылат. Бирок жакынкы чакыруулардын аракет этүү области чакырылуучу процедура баяндалган модулдун алкагы менен гана чектелет.

near жана far дирктивалары процедуранын/функциянын баяндоосунда анын операторлор блогунан мурда көрсөтүлөт.

```
Function SH(x:Real):Real; far;
    Begin
        SH:=(Exp(x)-Exp(-x)/2;
    End;
```

Көрсөтүлгөн чакыруу моделине жараша чыгуучу коддун генерациясын компилятордун {$F+} директивасы менен башкарууга болот. {$F+} абалында компиляцияланууучу процедуралар жана функциялар ар дайым чакыруунун алыскы тибине {far} ээ болушат, ал эми {$F-} абалында компилятор автоматтык турде жакынкы модельди тандап алат. Көрсөтүлбөгөн учурда (по умолчанию) {$F-} директивасы пайдаланылат.

10.5.2. forward директивасы

forward директивасы эки процедуранын өз ара рекурсиясы үчүн, ошондой эле тексттин структураланышын жакшыртуу жана жакшы окулушун жогорулатуу үчүн пайдаланууучу процедуралардын жана функциялардын озуп кетүүчү (опережающие) баяндамалары деп аталуучу баяндамалары үчүн пайдаланылат.

```
Program Recurs;
   uses Crt;
     procedure Rec1 (i: Byte ): forward;

     procedure Rec2 (i: Byte)
     begin
          WriteIn (‘ рекурсия’);
          Rec 1 (i)
     end;
     procedure Rec1;
```

```
 begin
      if i > 0  then
        begin
            Write (‘ Өз ара‘);
            Rec2(i-1)
        end
 end;
begin ClrScr;
Rec1 (5)
 end.
```

Жыйынтык:

```
Өз ара рекурсия
Өз ара рекурсия
Өз ара рекурсия
Өз ара рекурсия
Өз ара рекурсия
```

10.5.3. interrupt директивасы

interrupt директивасы үзгүлтүктөрдүн (прерывания) процедурасын баяндоо үчүн кызмат кылат. Үзгүлтүктөр процедурасынын бөркүндө параметрлер катары регистрлердин аттары баяндалган болушу керек.

```
Procedure intrpt(Flags,CS, IP, AX, BX, CX, DX, SI, DI, DS, ES, BP:
                                                    Word); interrupt;
```

Эгерде процедурадагы параметрлердин кээ бирлери пайдаланылбаса анда аларды жазбай таштап коюуга болот. Бирок параметрлердин тизмесинде регистирлердин аттарынын өз ара жайланышын өзгөртүүгө болбойт.

10.5.4. export директивасы

export директивасы процедураны же функцияны экспорттолуучу процедура же функция катары баяндайт. Бул директиваны жана экспорттолуучу процедураларды «Динамикалык байланылуучу библиотекалар (DLL)» темасында карап өттүк.

10.5.5. external директивасы

Бул директива өзүнчө компиляцияланган ассемблер тилинде жазылган процедураларды жана функцияларды байланыштыруу үчүн, ошондой эле динамикалык компоновкалануучу библиотекалардан

импорттолуучу процедураларды баяндоо үчүн арналган. Ассемблер процедуралар менен болгон байланыш - ассемблердик кыстырмаларга арналган бапта каралмакчы ал эми библиотекалардан импорттоо «Динамикалык байланылуучу библиотекалар (DLL)» темасында каралды.

10.5.6. assembler директивасы

assembler директивасы external директивасы сыяктуу эле ассемблер тилиндеги процедураларды баяндоо үчүн колдонулат. Айырмасы, accembler директивасы динамикалык компоновкаланууучу библиотекалардын баяндоолорунда пайдаланылбайт.

10.5.7. inline директивасы

Бул директива программистке комондаларды түздөн-түз машиналык коддо жазууга мүмкүнчулук берет.

Assembler жана inline директивалары ассемблердик кыстырмаларга арналган темада каралмакчы.

11. РЕКУРСИЯ

Убакыт учкул,
Уттурбоого умтул.
(Даанышман ойлор)

11.1. Рекурсия түшүнүгү жана негизги аныктоолор

Толук эмес түрдө (частично) кайра өзү аркылуу аныкталуучу объект рекурсивдик объект деп аталат.

Рекурсия идеялары адамдарга мурдатан эле белгилүү болгон. Рекурсивдик аныктолоор күчтүү аналитикалык аппарат катары илимдин көптөгөн тармактарында, өзгөчө математикада көп пайдаланылат.

n! факториал функциясын карап көрөлү. Эреже катары, аны алгачкы **n** бүтүн сандын көбөйтүндүсү катары аныкташат:

n! = 1 * 2 * 3 * 4 * . . . * n

Албетте, мындай көбөйтүндүнү итеративдик конструкциялардын бирөөсүнүн, мисалы, for циклинин жардамында жеңил эле эсептөөгө болот.

```
program factorial;
 var
    Fact:    Longint;
    n, i:    Integer;
  begin
    Write('n санын киргизгиле:');
    Readln(n);
    Fact: =1;
    For  i := 1 to  n  do  Fact := Fact * i;
    Writeln(' n! факториал=', Fact);
  end.
```

Бирок факториалдын рекуренттик формула пайдаланылган төмөнкүдөй көрүнүшкө ээ болгон башка да аныктамасы бар:

(1) 0! = 1
(2) $\forall$ n > 0 үчүн n! = n*(n-1)!

Рекуренттик формулалардын жардамында болгон аныктоолорду кээде рекурсивдик аныктоолор деп аташат. Факториал үчүн биринчи (итеративдик) аныктама жөнөкөй көрүнүшү мүмкүн болсо, анда Фибоначчи сандары үчүн төмөнкү рекурсивдик аныктама

(1) F(1)=1,
(2) F(2)=1,
(3) $\forall$ n>2 үчүн f(n)= F(n-1) + F(n-2)

түздөн-түз эсептөөнүн төмөнкү формуласына караганда

$$F(n)=\frac{\left(\frac{1+\sqrt{5}}{2}\right)^n-\left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

бир кыйла жакшы көрүнөт.

Буга окшогон рекуренттик формулалар башка көптөгөн математикалык аныктамалар үчүн да бар. Албетте, рекурсияны пайдаланбай туруп деле рекуренттик фомула боюнча эсептөөнү уюштуруу мүмкүн экендиги түшүнүктүү. Бирок анда программанын сапаты жана анын баштапкы формулага эквиваленттүү экендигин далилдөө маселеси пайда болот. Рекурсияны пайдалануу рекуренттик формулалар боюнча эсептөөлөрдү жеңил (дээрлик сөзмө-сөз) программалоо мүмкүнчүлүгүн берет. Мисалы, ***n!*** факториалды эсептөө үчүн рекурсивдик функцияны пайдалануу менен түзүлгөн программа төмөнкүдөй көрүнүштө болот.

```
program    Factorial;
 var
      n: Integer ;
 function  Fact( i : Integer): Longint;
 begin
      if  i =1 then  Fact := 1
            else    Fact := i* Fact ( i-1 )
 end;

 begin
      Write('n санын киргизгиле:');
      Readln(n);
      Writeln('n! факториал=' , Fact(n));
  end.
```

Рекурсивдик аныктаманын мазмууну жана анын күчү, ошондой эле анын эң башкы арналышы мына мында турат: ал чектүү туюнтманын жардамы менен объекттердин чексиз көптүгүн аныктоого мүмкүнчүлүк берет. Ушуга эле окшош чектүү рекурсивдик алгоритмдин жардамы менен чексиз эсептөөнү аныктоо мүмкүн, болгондо да андай алгоритм тексттин фрагменттеринин кайталанышын кармап турбаган алгоритм болот.

Рекурсивдик алгоритимдерди түзүү үчүн процедура же функция түшүнүгүнүн болушу зарыл жана жетиштүү. Бул нерсе, процедуралардын жана функциялардын аракеттердин

(операторлордун) каалагандай удалаштыгына ат берүүгө жана кийин ал аттын жардамында аракеттердин бул удалаштыгын чакырууга мүмкүнчүлүк беришинен келип чыгат.

Мисалы, төмөнкү процедура бардыгыбызга белгилүү болгон саптарды чексиз печаттайт.

```
program  EndLess1;
   procedure  BoyAndDog1;
   begin
       Writeln('Мойногум эй,  мойногум');
       Writeln('Ээрчигенди   койбодуң');
       Writeln('Сени көрсөм, менин да');
       Writeln('Келе берет ойногум');
      BoyAndDog1
    end;
     begin
         BoyAndDog1;
     end.
```

Бирок, эгерде процедураны чакыруу операторун текстти чыгаруудан мурда койсок, төмөнкүдөй кылып,

```
program  EndLess2;
      procedure BoyAndDog2;
      begin
          BoyAndDog2;
          Writeln('Мойногум эй,  мойногум');
          Writeln('Ээрчигенди   койбодуң');
          Writeln('Сени көрсөм, менин да');
          Writeln('Келе берет ойногум');
     end;
     begin
       BoyAndDog2;
     end.
```

анда бул программа да чексиз иштегени менен эч нерсени печатка чыгарбайт. Бул, процедураны чакыруу операторунун биринчи турушу жана буга тиешелүү түрдө BoyAndDog2 процедурасынын жаңы көчүрмөсүн (копиясын) чакыруу текстти чыгаруудан мурда болуп өтүшү менен түшүндүрүлөт. Процедуранын кийинки көчүрмөсүндө дагы эле ушуга окшош аракеттер жана ушул сыяктуу улана берет. Натыйжада процедурада чыгаруу оператору турганы менен кандайдыр бир текстти чыгарбастан эле BoyAndDog2 процедурасын чексиз чакыруу жүргүзүлө берет. Чындыгында компьютерде мындай

чексиз чакырууну аткаруу стектин ашып кетишине (переполнение) жана аткаруу убактысынын катасынын пайда болушуна алып келет.

Рекурсивдик процедуралар пайдаланылган программалар жөнөкөйлүгү, көрсөтмөлүүлүгү жана текстинин компакттуулугу менен айырмаланышат.

Рекурсивдик алгоритмдердин мындай сапаттары төмөндөгүлөрдөн улам келип чыгат: рекурсивдик процедура эмне кылуу керек экендигин көрсөтөт, ал эми рекурсивдик эмес процедура кантип жасоо керек экендигине көбүрөөк басым жасайт.

Бирок, программанын жөнөкөйлүгү үчүн оперативдик эсти экономдуу эмес пайдалануу аркылуу төлөөгө туура келет, анткени рекурсивдик процедуралар рекурсивдик эмес процедураларга караганда алда канча көп өлчөмдөгү оперативдик эсти талап кылат. Ар бир рекурсивдик чакырууда локалдык өзгөрүлмөлөр жана ошондой эле процедуранын параметрлери үчүн эстин жаңы ячейкалары ажыратылып турат.

Ошентип кандайдыр бир A локалдык өзгөрүлмөсүнө рекурсиянын ар түрдүү деңгээлдеринде ар түрдүү маанилерди кармап турушу мүмкүн болгон эстин ар түрдүү ячейкалары тиешелеш келет.

Ошондуктан рекурсиянын і-деңгээлдеги A өзгөрүлмөсүнүн маанисинен ушул і-деңгээлде труп гана пайдалануга болот.

Рекурсияга тиешелүү болгон кээ бир аныктоолорду келтиребиз.

Программа аткарылып жаткан кездеги процедураны кайтарымсыз рекурсивдик чакыруулардын максималдык саны *рекурсиянын тереңдиги* деп аталат.

Убакыттын ар бир конкреттүү моментиндеги рекурсивдик чакыруулардын саны *рекурсиянын учурдагы деңгээли* деп аталат.

11.2. Рекурсивдик процедуралардын формалары

Жалпы учурда каалагандай Rec рекурсивдик процедурасы опера-торлордун кандайдыр бир S көптүгүн жана бир же бир нече P рекурсивдик чакыруу операторлорун өз ичине камтып турат.

Биз жогоруда карагандай, шартсыз рекурсивдик процедуралар чексиз процесстерге алып келет, ошондуктан дагы бул проблемага өзгөчө көңүл бөлүү керек, анткени практика жүзүндө чексиз өзүн

чакыруучу процедураларды пайдалануу мүмкүн эмес. Мындай мүмкүн эместик төмөнкүдөн келип чыгат: рекурсивдик процедуранын ар бир көчүрмөсүнө эстин кошумча областын ажыратуу зарыл болот, а чексиз эс деген жашабайт.

Ошентип, рекурсивдик процедураларга коюлган башкы талап – рекурсивдик процедуранын чакырылышы рекурсиянын кайсы бир деңгээлинде жалган болуп кала тургандай шарт боюнча аткарылышы керек.

Эгерде шарт чын болсо, анда рекурсивдик ылдыйлоо (спуск) улантылат. Качан ал жалган болуп калганда ылдыйлоо бүтөт жана берилген убактагы бардык чакырылган рекурсивдик процедуранын көчүрмөлөрүнөн кезек менен рекурсивдик кайтуу(возврат) башталат.

Рекурсивдик процедуранын структурасы ар түрдүү үч форманы кабыл алышы мүмкүн:

1. Рекурсивдик чакырууга чейин аркеттерди аткаруу менен болгон форма (рекурсивдик ылдыйлоодо аракеттердин аткарылышы менен болгон).

```
Procedure  Rec;
 begin
    S;
    if  шарт  then   Rec;
  end;
```

2. Аракеттердин аткарылышы рекурсивдик чакыруудан кийин болгон форма (аракеттердин аткарылышы рекурсивдик кайтууда (возврат) болгон).

```
procedure  Rec;
     begin
       if   шарт  then   Rec;
       S;
     end;
```

3. Аракеттердин аткарылышы рекурсивдик чакырууга чейин да, чакыруудан кийин да болгон форма (рекурсивдик ылдыйлоодо да, рекурсивдик кайтууда да аракеттердин аткарылышы менен).

```
procedure   Rec;
    begin
      S1;
       if  шарт  then   Rec;
      S2;
    end;
```

же

```
procedure   Rec;
    begin
      S1;
       if  шарт  then   Rec;
      S2 ;
     end;
```

Рекурсивдик процедуралардын бардык формаларынын практикалык колдонулуштары бар. Көптөгөн маселер үчүн, анын ичинде факториалды эсептөө үчүн да, рекурсивдик процедуранын кайсы формасы пайдаланылышы маанига ээ эмес.

Бирок, маселелердин ушундай бир классы бар, аларды чечүү программисттен рекурсивдик процедуралардын жана функциялардын ишинин жүрүшүн аң-сезимдүү түрдө башкарууну талап кылат. Айрым учур катары мындай маселелер болуп, тизмелик жана дарак сымал сруктурадагы берилгендерди пайдалануучу маселелер эсептелет.

Мисалы, транслятордуталдап иштеп чыгууда талдоонун атрибутташтырылган даректары деп аталган сруктура кеңири колдонулат. Алар менен иштөө програмисттен рекурсиянын жүрүшүн аң-сезимдүү түрдө багыттай билүүнү талап кылат: кээ бир аракеттерди ылдыйлоодо гана аткаруу мүмкүн, ал эми башкаларын – кайтуу да гана. Ошон үчүн **рекурсивдик механизмди терең түшүнүү жана аны өзү каалагандай башкара билүү квалификациялуу программист үчүн зарыл сапат болуп саналат**.

Рекурсивдик камтылуучу программалардын (подпрограмм) биринчи эки формасын факториалды (n!) эсептөөнүн мисалында, үчүнчүсүн - кийрилген жолчону рекурсивдик печаттонун мисалында карайбыз.

11.2.1. Рекурсивдик ылдыйлоодо аракеттерди аткаруу

Рекурсивдик ылдыйлоодо иштөөчү факториалды эсептөөнүн уникалдык алгоритмин реализациялоо үчүн рекурсивдик функцияга кошумча эки параметрди киргизүү талап кылынат:

Mult – рекурсивдик чакырууга чейин (б.а. ылдыйлоодо) факториалдын топтолгон маанисин кезектеги көбөйтүүчүгө көбөйтүү амалын аткаруу үчүн.

m – рекурсивдик функциянын конкреттүү глобалдык өзгөрүлмөнүн атынан көз каранды болбостугун камсыз кылуу, б.а. функциянын уникалдуулугун көтөрүү үчүн.

Эсептөөнү ылдыйлоодо аткаруучу Fact_Dn рекурсивдик функциясын пайдаланган Factorial_Down программасы төмөнкү көрүнүштө болот.

```
program Factorial_Down;
  var   n : Integer;
  function Fact_Dn ( Mult : Longint ;  i , m : Integer) : Longint;
 begin
   Mult := Mult * i; {факториалдын топтолушу  рекурсивдик }
         { чакыруу операторуна чейин турат.                 }
         { Ошон үчүн эсептөө ылдыйлоодо                     }
         { аткарылат.                                       }

 if  i =  m   then  Fact_Dn  := Mult
                    else   Fact_Dn  :=  Fact_Dn( Mult , i+1, m)
  end ;

 begin
        Write (‘ n  санын киргизгиле.’);
        Readln ( n ) ;
        Writeln(‘  n!  факториалы = ’, Fact_Dn(1, 1, n));
   end.
```

Fact_Dn функциясы тарабынан аткарылуучу аракеттерди демонстрациялоо үчүн рекурсиянын деңгээлдери боюнча анын параметлеринин маанилерин трассирлөө (трссировка) жадыбалын келтиребиз. Бул жадыбалда n=5 болгон конкреттүү учур каралган.

Рекурсиянын учурдагы деңгээли		**Рекурсивдик ылдыйлоо**			**Рекурсивдик кайтуу**	
0		**Кийрүү (n=5)**		**Fact_Dn (1,1,5)**	**Чыгаруу: n!=120**	
1		**Mult:=1*1 (1) ;**	**I=1;**	**Fact_Dn (1,2,5)**	**Fact_Dn :=120;**	
2		**Mult:=1*2 (2);**	**I=2;**	**Fact_Dn (2,3,5)**	**Fact_Dn :=120;**	
3		**Mult:=2*3 (6) ;**	**I=3;**	**Fact_Dn (6,4,5)**	**Fact_Dn :=120;**	
4		**Mult:=6*4 (24) ;**	**I=4;**	**Fact_Dn (24,5,5)**	**Fact_Dn :=120;**	
5		**Mult:=24*5 (120) ;**	**I=5;**	**Fact_Dn:=120;**	**Fact_Dn :=120;**	

11.2.2. Рекурсивдик кайтууда аракеттердин аткарылышы

Биз ушул баптын (главанын) башталышында карап өткөн Fact рекурсивдик функциясын пайдалануучу Factorial программасы факториалды рекурсивдик кайтууда эсептейт. Бирок, бул өтө эле түшүнүктүү жана көрүнүктүү эмес, анткени ал жерде Fact функциясында рекурсивдик чакыруу жана көбөйтүү амалы бир эле

ыйгаруу операторуна бириктирилген. Рекурсивдик кайтуудагы жүргүзүлүүчү ишти бир кыйла түшүнүкүтүү кылып демонстрациялоо үчүн рекурсивдик чакыруу жана факториалды топтоо оператору бири биринен айкын түрдө ажыратылган Fact_Up программасын келтиребиз.

```
program Factorial_Up;
     var         n : Integer;
   function Fact_Up ( i : Integer): Longint;
     var     Mult :  Longint;

   begin
     if  i = 1  then  Mult := 1
           else Mult:=Fact_Up(i-1);
    Fact_Up:=Mult * i        {Факториалдын топтолушу рекурсивдик}
        { чакыруу операторунан кийин турат. Анда, демек}
        {эсептөө рекурсивдик кайтууда аткарылат.                    }
   end;
   begin
            Write('n санын киргизгиле:');
            Readln(n);
            Writeln('n! факториал = ' ,  Fact_Up(1, 1 , n));
    end.
```

Fact_Up функциясы үчүн да рекурсиянын деңгээлдери боюнча трассирлөө жадыбалын келтиребиз.

Рекурсия-нын учур-дагы дең гээли	**Рекурсивдик ылдыйлоо**	**Рекурсивдик Кайтуу**
0	**Кийрүү (n=5); Fact_Up (5)**	**Чыгаруу: n!=120**
1	**i =5; Mult:= Fact_Up (4);**	**Fact_Up:=24*5 (120);**
2	**i =4; Mult:= Fact_Up (3);**	**Fact_Up:=6*4 (24);**
3	**i =3; Mult:= Fact_Up (2);**	**Fact_Up:=2*3 (6);**
4	**i =2; Mult:= Fact_Up (1);**	**Fact_Up:=1*2 (2);**
5	**i =1; Mult:=1;**	**Fact_Up:=1*1 (1);**

11.2.3. Рекурсивдик ылдыйлоодо да, кайтууда да аракеттердин аткарылышы

Рекурсивдик камтылуучу программалардын үчүнчү формасын төмөнкү маселенин жардамында көрсөтөбүз.

Маселе. Киргизилген 'SALAM' жолчосунун символдорун тескери багытта печатка чыгаргыла.

Бул маселенин чечилиши Reverse рекурсивдик процедурасын пайдаланууучу төмөнкү Reverse_String программасы көрүнүшүндө аткарылган.

Eoln функциясынын мааниси, эгер жолчо али бүтө элек болсо False, ал эми жолчонун акыркы символу окулганда True болорун эске алабыз.

```
program  Reverse_String;
   procedure     Reverse;
   var     Ch :     Char;
   begin
     if    not    eoln    then
      begin
            Read(Ch);
            Reverse;
           Write(Ch);
       end
  end;
 begin
    Reverse
 end.
```

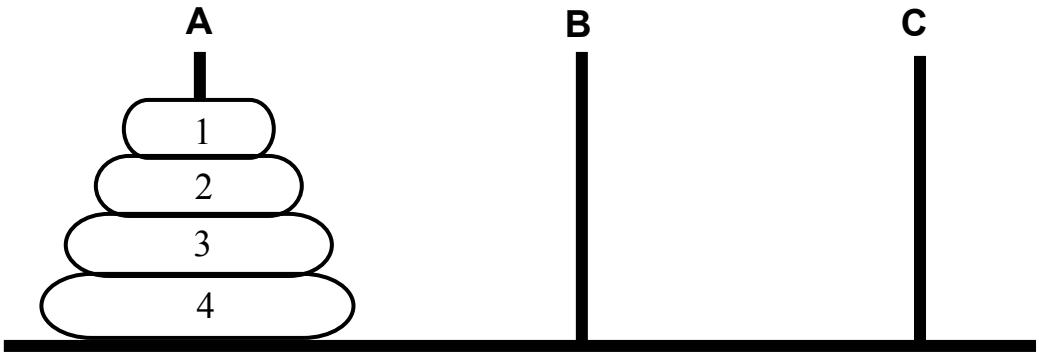
Эгерде программаны ишке салгандан кийин кийрилүүчү жолчо катары ‘SALAM’ сөзүн киргизсек, анда мындай баштапкы жолчого тиешелүү трассирлөө жадыбалы, рекурсиянын деңгээлдери боюнча төмөнкү көрүнүштө болот.

Рекурсиянын учурдагы деңгээли	**Рекурсивдик ылдыйлоо**	**Рекурсивдик кайтуу**
0	**Reverse;**	
1	**eoln=False; Кийрүү: ‘S’ ; Reverse ;**	**Чыгаруу :’S’;**
2	**eoln=False; Кийрүү: ‘A’ ; Reverse ;**	**Чыгаруу :’A’;**
3	**eoln=False; Кийрүү: ‘L’ ; Reverse ;**	**Чыгаруу :’L’;**
4	**eoln=False; Кийрүү: ‘A’ ; Reverse ;**	**Чыгаруу :’A’;**
5	**eoln=False; Кийрүү: ‘M’ ; Reverse ;**	**Чыгаруу :’M’**
6	**Eoln=True ;**	

11.3. Ханой мунаралары жөнүндөгү маселе

Бир кыйла татаал рекурсивдик байланыштардын алгоритмдерде чагылдырылышын Ханой мунаралары жөнүндөгү маселенин мисалында көрсөтөбүз.

Маселе. Үч А, Б, С – мамычалары берилген. А мамычасында ар түрдүү диаметрдеги, жогортон төмөн карай номерленген төрт диск ар бир кичине диск чоң дисктин үстүндө боло тургандай кылып жайлаштырылган.



Ушул төрт дискти, алардын өз ара жайланышы сактала тургандай кылып С мамычасына которуу керек. В мамычасын жардамчы мамыча катары пайдаланууга уруксат берилет. Маселени чечүүдө бир кадам менен кандайдыр бир мамычадагы эң жогорку дисктердин бирөөсүн гана жылдырууга уруксат. Андан сырткары чоң дискти эч качан диаметри боюнча кичине болгон дисктин үстүнө жайгаштырууга болбойт.

Бул коюлган маселени чечүүгө кандай ыкма менен киришүүнү аныктоо үчүн **n** диск менен болгон бир кыйла жалпы учурду карайбыз. Эгерде биз **n** диск үчүн болгон маселенин чечимин **n-1** диск үчүн чечимдин терминдеринде таап алсак, анда коюлган маселе чечилген болоор эле, анткени **n-1** диск үчүн маселени **n-2** диск менен болгон маселенин чечиминин терминдеринде чечүүгө келтирүүгө болот жана ушул сыяктуу тривиалдык учур болгон бир диск болгон учурга чейин жетүүгө болот. Бир диск **(n=1)** болгон учур үчүн чечим элементардык болуп калат. Мында жөн гана бир дискти А мамычасынан С мамычасына которуу керек болот.

Мына ошентип, эгер **n** диск үчүн чечимди **n-1** дисктин терминдеринде формулировкаласак, анда иш жүзүндө рекурсивдик процедуранын алгоритмин алабыз жана ал алгоритмдин жардамында коюлган маселенин максатына жетүү жеңил эле болот. Ушундай алгоритмдин оозэки баяндалышын карайбыз.

if n = 1

(then)

1. Бул жалгыз дискти A мамычасынан C мамычасына котор жана токто.

(else)

1. Жогорку n-1 дискти A мамычасынан B мамычасына, C мамычасын жардамчы мамыча катары пайдаланып котор.
2. Калып калган тмөнкү дискти A мамычасынан C мамычасына котор.
3. B мамычасынан n-1 дискти C мамиычасына, A мамычасын жардамчы мамыча катары пайдаланып котор.

Аракеттердин мындай удаалаштыгы каалагандай **n** мааниси үчүн корректүү чечимди берет деп ишенимдүү айтууга болот. Бул мына мындан келип чыгат. Эгерде **n=1** болсо, анда корректүүлүк көрүнүп турат. Эгер **n=2** болсо, анда **1** ге барабар болгон **(n-1)** диск учуру үчүн чечимге эбак ээ болгон болобуз. Ушуга эле окшош эгер **n=3** болсо, анда **2** ге барабар болгон **(n-1)** диск учуру үчүн чечимге ээ болобуз. Ушуга окшош бул чечим **n**=1,2,3,4. . . жана башка кандайдыр бир башка **n** үчүн иштейт.

Коюлган маселени Move_Disks рекурсивдик процедурасынын жардамында чечүүчү Hanoi_Towers программасын келтиребиз.

```
program Hanoi_Towers;
  uses Crt;
  var   n:  Integer;
  procedure Move_Disks (n: Byte ; Source, Dest, Tmp; Char );
    { n -Source мамычасындагы дисктердин саны}
    { Source – баштапкы мамыча }
    { Dest  -   дисктерди котору керек болтон мамыча}
    {Tmp  –  жардамчы мамыча }
 begin
        if  n = 1  then
           writeln(‘ 1-номердеги дискти’ ,Source, ‘мамычасынан’,
                     Dest, ‘мамычасына жылдыргыла’ );
              else  begin
          { Которуу керек болгон дискти аралык диск катары      }
          { пайдалануу менен жогорку n-1 дисктерди баштапкы }
          { мамычадан жардамчы мамычага которобуз              }
               Move _Disks (n-1, Source, Tmp, Dest);
               Writeln (n:1, ‘номердеги дискти’,
 Source,’мамычасынан’,  Dest,’мамычасына жылдыргыла’);
         {Баштапкы дискти аралык диск катары пайдаланып}
```

```
    {жардамчы мамычадагы бардык n-1 дискти, керек }
    { болгон мамычага жылдырабыз}
        Move _ Disks (n-1, Tmp, Dest, Source);
          end
end ;
begin
        ClrScr;
        Write('Дисктердин санын киргизгиле:');
        Readln(n);
        Writeln;
        Writeln('Маселени чечүү үчүн инструкциялардын
                                удаалаштыгы:');
        Writeln;
        Move _ Disks (n,  'A',  'C',  'B' );
end.
```

А мамычасындагы баштапкы дисктердин саны **4** кө барабар болгон учурда Hanoi_Towers программасынын ишинин жыйынтыгы төмөндөгүдөй болот :

```
Дисктердин санын киргизгиле: 4
        Маселени чечүү үчүн инструкциялардын удаалаштыгы:

1-      номердеги дискти А мамычасынан В мамычасына жылдыргыла
2-      номердеги дискти А мамычасынан С мамычасына жылдыргыла
1-      номердеги дискти В мамычасынан С мамычасына жылдыргыла
2-      номердеги дискти А мамычасынан В мамычасына жылдыргыла
1-      номердеги дискти С мамычасынан А мамычасына жылдыргыла
2-      номердеги дискти С мамычасынан В мамычасына жылдыргыла
1-      номердеги дискти А мамычасынан В мамычасына жылдыргыла
4 -     номердеги дискти А мамычасынан С мамычасына жылдыргыла
1-      номердеги дискти В мамычасынан С мамычасына жылдыргыла
2-      номердеги дискти В мамычасынан А мамычасына жылдыргыла
1-      номердеги дискти С мамычасынан А мамычасына жылдыргыла
3-      номердеги дискти В мамычасынан С мамычасына жылдыргыла
1-      номердеги дискти А мамычасынан В мамычасына жылдыргыла
2-      номердеги дискти А мамычасынан С мамычасына жылдыргыла
1-      номердеги дискти В мамычасынан С мамычасына жылдыргыла
```

Каралган маселени жыйынтыктап жатып математикалык индукция усулу менен программалоонун рекурсивдик усулунун ортосундагы аналогияны белгилеп кетүүгө болот. Чындыгында эле, алгач бир диск үчүн маселенин чечими формулировкаланды эле. Кийин **(n-1)** диск үчүн маселенин чечими жашайт деп алынып, ошонун негизинде өзүн-өзү чакыруучу **n** дискти жылдыруу процедурасы тургузулду. Ушул көрсөтүлгөн аналогия

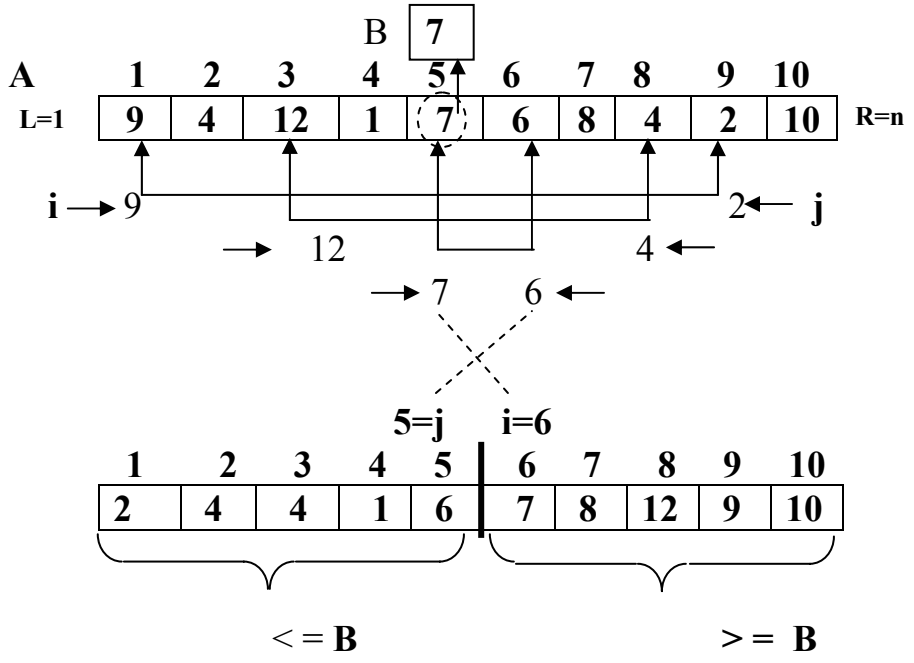
программалоонун рекурсивдик усулдарынын күчтүүлүгүн көрсөтөт, анткени алардын жардамында рекурсивдик формулалар боюнча тургузулган алгоритмдерди, ошондой эле Ханой мунаралары жөнүндөгү маселеге окшогон жалпы мүнөздөгү алгоритмдерди табигый стилде программалоого болот.

11.4. Тез сорттоо

Массивди тез сорттоо усулунун реализациясы, рекурсияны пайдалануунун эң сонун мисалы болот. Бул усул 1962-жылы Оксфорд университетинин профессору К.Хоар тарабынан иштелип чыгылган.

Усулдун принциби:

1. А массивинин борбордук элементин тандап алабыз жана аны B өзгөрүлмүсүнө жазабыз. Андан кийин массивдин элементтери кезек менен солдон-оңго карай жана оңдон-солго карай каралып чыгылат. Солдон оңду карай жүргөн учурда B элементинен чоң же ага барабар болгон A[i] элементин издейбиз да анын позициясын эске сактап коебуз. Ал эми оңдон солду көздөй жүргөндө B элементинен кичине же ага барабар болгон A[j] элементин издейбиз, анын да позициясын эске сактап коебуз. Табылган элементтердин орундарын алмаштырабыз жана коюлган шарттар боюнча дагы эки жактуу издөөнү улантабыз. Табылган элементтердин орундарын дагы алмаштырабыз жана ушул сыяктуу бул процессти издөөнүн кезектеги итерациясында эки тараптуу i жана j индекстери кесилишип калмайынча уланта беребиз.



Ушуну менен биринчи этап бүттү деп эсептелет. Ушундан кийин баштапкы массивдин элементтери B маанисине салыштырмалуу бардык маанилери B дан кичине же барабар болгон, алар i жана j индекстеринин кесилиш чек арасынын сол жагында жатат, жана бардык элементтери B дан чоң жана барабар болгон, алар кесилиш чек арасынын оң жагында жатат жана эки бөлүккө бөлүнүп калат. Ошентип B маанисине салыштырмалуу берилген массив сорттолгон болуп калат, бирок анын оң жана сол бөлүктөрү али сорттоло элек.

2. Экинчи этапта биринчи этаптагы аракеттер массивдин оң жана сол бөлүктөрү үчүн өз өзүнчө аткарылат. Натыйжада, массив бири бири менен сорттоо боюнча кесилишпеген, ар бирин өзүнчө иреттөөгө мүмкүн болгон төрт бөлүккө бөлүнүп калат. Үчүнчү этапта биринчи этаптагы аракеттер төрт бөлүктүн ар бири үчүн өзүнчө аткарылат, жана ушул сыяктуу сорттолуучу бөлүктүн узундугу качан бир элементке барабар болуп калмайынча улантыла берет, анда демек массивдин бардык элементтери иреттелип калган болот.

Ар бир этапта бирдей эле аракеттер кайталана бергендиктен (бирок массивдин ар түрдүү индекстик аймактарында), анда аларды өзүн өзү чакыруучу рекурсивдик процедура көрүнүшүндө жазуу ыңгайлуу. Мындай процедура төмөнкү программада пайдаланылган:

```
program QuickSort;
uses Crt;
const
      n = 20;
      var
         A := array [1..n] of  Integer;
          i : Word;

      procedure Qsort ( L , R : Word);
      var
         B, Tmp :  Integer;
          i, j: Word;
            begin
             B:=A[ (L+R) div 2];
             i: =L ;  j:= R;

               While I <=  j  do
               begin
                    While  A[i] < B  do  i := i+1;
                    While  A[j] > B  do  j := j+1;
                     If  i <= j  then
```

```
                    begin
                        Tmp :=  A[i];
                        A[i] := A [j];
                        A[j] := Tmp;
                         i := i+1;
                  j := j+1;
              end;
           end;
                if  L<j then Qsort (L, j );
                if  i<R then Qsort (i, R );
             end;

begin
       ClrScr;
       Writeln('Массивдин элементтерин киргизгиле');
       for i :=1 to n do Read( A[ i ]);  Readln;
       {------------------------------ - - - - - - - - - - - - - - - - - -}
       Qsort (1, n);
       {------------------------------ - - - - - - - - - -- - - - - - - - }
       Writeln('Сорттолгон массив:');
       for i :=1 to n  Write (A[ i ]:8);
       Writeln;
    end.
```

12. ФАЙЛДАР

Талбай алсаң билимди,
Тазартасың дилиңди.
(Даанышман ойлор)

12.1. Физикалык жана логикалык файл түшүнүгү

Файл түшүнүгүнүн эки жагы бар. Бир жагынан файл – кандайдыр бир информацияны кармап турган сырткы эстин ысымдашкан областы. Файлды мындайча түшүнүүдө аны ***физикалык файл*** деп аташат, б.а. ал информацияларды ташып жүрүүчү кандайдыр бир материалдык нерседе физикалык түрдө жашайт.

Экинчи жактан, файл – бул программалоодо пайдаланылуучу көптөгөн структалардын бири. Файлды мындайча түшүнүүдө аны ***логикалык файл*** деп аташат, б.а. программаларды жазууда биздин логикалык элестетүүбүздө гана жашайт. Программаларда логикалык файлдар анык бир типтеги файлдык өзгөрүлмөлөр аркылуу көрсөтүлөт.

12.1.1. Физикалык файлдын структурасы

Физикалык файлдын структурасы информацияларды ташып жүрүүчүнүн – катуу магниттик дисктин (КМД) же ийилчек магниттик дисктин – эсинин байттарынын жөнөкөй удаалаштыгы болуп эсептелет.

байт	байт	байт	….	байт	байт	байт

12.1.2. Логикалык файлдын структурасы

Логикалык файлдын структурасы – бул программада файлды кабыл алуу жолу болуп саналат. Образдуу айтканда бул биз файлдын физикалык структурасын көрө турган «шаблон» («терезе»). Программалоо тилдеринде мындай «шаблондорго» файлдардын компоненттери катары уруксат берилүүчү берилгендердин типтери тиешелеш келет. Turbo Pascal тилинин «шаблондорунун» кээ бирөөлөрүн образдуу элестетүү төмөнкү сүрөттөрдө келтирилген.

File of byte

байт	байт	байт	байт	….	байт	Eof

File of char

Символдун коду	Символдун коду	Символдун коду	…	Символдун коду	Eof

File of Integer

Белгиге ээ болгон бүтүн	Белгиге ээ болгон бүтүн	…	Белгиге ээ болгон бүтүн	Eof

File of T, мында T=record
a: byte;
в: char;
c: integer;
end;

Байт	Сим-волдун коду	Сим-волдун коду	…	Байт	Сим-волдун коду	Белгиге ээ болгон бүтүн	Eof

Пайдаланылыш принциби боюнча файлдын логикалык структурасы массивдин структурасына абдан окшош. Массивдер менен файлдардын ортосундагы айырмачылыктар төмөндөгүлөрдө.

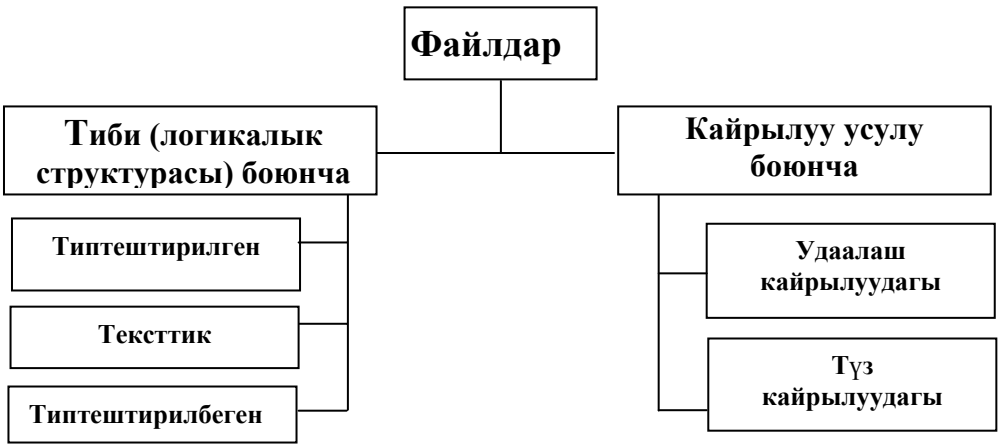
Массивде элементтердин саны эсти бөлүштүрүү моментинде фиксирленип ал бүтүндөй оперативдик эсте жайгашат. Массивдин элементтерин номерлөө аны жарыялоо кезинде көрсөтүлгөн төмөнкү жана жогорку чек араларга тиешелеш аткарылат.

Файлда анын элементтеринин саны программанын иштөө процессинде өзгөрүп кетиши мүмкүн жана ал тышкы информацияларды ташуучуларда жайгашат. Файлдын элементин номерлөө (текстик файлдардан башка) нөлдөн баштап солдон оңду карай жүгүзүлөт да ал элементтердин саны убакыттын ар бир моментинде белгисиз. Бирок, файлдын аяк жагында файлдын бүтүшүнүн атайын белгиси Eof символу жайгашып, мындай символ катары ASCII нин коду 26 га барабар болгон (Ctrl+Z) башкаруучу символу пайдаланылат. Андан сырткары, файлдар менен иштөө үчүн арналган стандарттык процедуралардын жана функциялардын жардамында файлдын узундугун аныктоо жана башка көп талап кылынуучу амалдарды аткаруу мүмкүн.

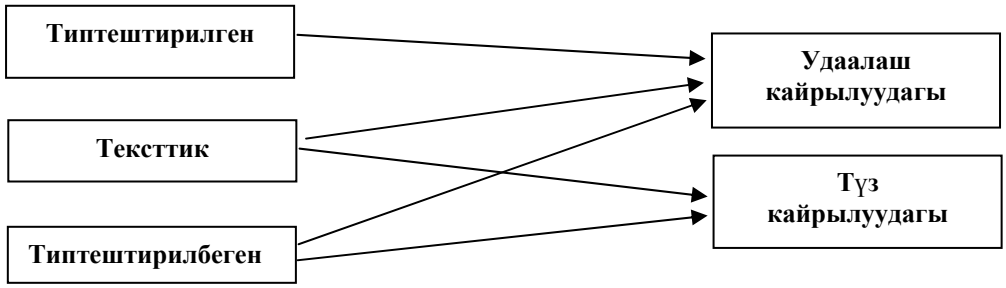
12.2. Turbo Pascal чөйрөсүндө файлдарды классификациялоо

Файлдар эки белги боюнча классификацияланат:

- тиби (логикалык структурасы) боюнча;
- файлдын элементтерине кайрылуу усулу боюнча.



Тиби боюнча ар түрдүү файлдарга кайрылуу усулун колдонуу мүмкүнчүлүгүн төмөнкү схемада көрсөтөбүз.



12.3. Файлдардын арналышы. Файлдарды ачуу жана жабуу

Катуу же ийилчээк магнитик дискте жайгашкан кандайдыр бир файл менен иштөө үчүн алгач ал файлды ушул физикалык файлга кайрылууга мүмкүнчүлүк берүүчү файлдык өзгөрүлмө (логикалык файл) менен байланыштырып алуу керек. Логикалык жана физикалык файлдарды байланыштыруу жабык (ачыла элек) файл үчүн гана пайдаланылышы мүмкүн болгон Assign процедурасы аркылуу аткарылат. Бул процедуранын биринчи параметри болуп файлдык өзгөрүлмө, ал эми экинчи параметри болуп маниси MS DOS то идентификаторлорду жазуу эрежеси боюнча көрсөтүлгөн физикалык файлдын аты болушу керек болгон жолчолук константа же жолчолук өзгөрүлмөнүн идентификатору эсептелет.

```
Assign (f, 'MyFile.Dat');
```

Бул келтирилген мисалда f логикалык файлын MyFile.Dat физикалык файлы менен ал MS DOS тун активдүү дискинин учурдагы каталогунда жайгашкан деген шартта байланыштырат. Эгерде Assign процедурасынын аракети MS DOS тун учурдагы түзүлүштөрүнөн көз каранды болбосун деп талап кылынса, анда дискти, каталогдордун жолун жана файлдын атын көрсөтүү менен файлдын толук аты жазылат, мисалы

```
Name: = ‘a:\MyFiles\MyFile.Dat’;
Assign (f,Name);
```

Файлдарда кандайдыр бир окуу жана жазуу амалдарын аткаруу үчүн оболу бул файлдар ачылган болушу керек.

Файлды ачуу Reset жана Rewrite процедуралары менен, а жабуу – Close процедурасынын жардамында аткарылат.

```
Reset(f);
Rewrite(f);
Close(f);
```

Reset процедурасы f файлдык өзгөрүлмөсү менен байланыштырылган бар болгон (сушествующий) физикалык файлды ачат. Эгер f – текстик файл болсо, анда элементтери удаалаш кайрылуу учурунда аны окууга гана болот, эгер f – типтештирилген файл болсо, анда ал удаалаш кайрылууда да, түз кайрылууда да аны окуу жана жазуу үчүн ачылган болот. Ачкан учурда файлдын учурдагы позициясынын көрсөткүчү анын башталышына коюлат.

Эгер көрсөтүлгөн аттагы физикалык файл жок болуп калса, анда аткаруу убактысынын катасы (ошибка времени исполнения) пайда болот, аны компилятордун {$I-} директивасын алып салуу менен жоготууга (подавить) болот. Директиваны ушундайча коюу менен IOResult функциясынын жардамында файлды ачуу амалынын аяктоо жыйынтыгын анализдөө мүмкүн. Бул функция, эгер амал жакшы аяктаган болсо 0 деген маанини, андай болбогондо катанын нөл эмес кодун берет.

Rewrite процедурасы, аты f файлдык өзгөрүлмөсү менен байланышылган жаңы физикалык файлды түзөт. Эгер мындай физикалык файл мурда эле бар болгон болсо, анда ал өчүрүлөт да анын ордуна жаңы бош файл түзүлөт. Файлды ачканда учурдагы позициянын көрсөткүчү файлдын башталышына коюлат.

Дээрлик бардык программаларда пайдалануучу дагы бир функция Eof функциясы.

```
Eof(f);
```

Eof функциясы True маанисин берет, эгер файлда учурдагы позициянын көрсөткүчү файлдын акыркы элементинде турса же файл бош болсо. Андай болбогондо False маанисин берет.

Мисал катары MyFile.Dat файлынын элементтеринин суммасын эсептөө маселесин карап көрөбүз.

```
Program FSumma;
 uses Crt;
 var
     f: file of integer;
     x: integer;
     Summa: LongInt;
  begin
      ClrScr;
     { $I-}
     Assign(f, 'MyFile.Dat');
     Reset(f);
      {I+}
     if   IOResult <>  0 then
         begin
                Writeln('файлды ачуу катасы');
        Halt(1);
         end;
                {Сумманы эсептөө}
         Summa:=0;
              While  not Eof(f)  do
           begin
               Read (f,x);
                Summa:= Summa+x;
 end;
 Writeln('Файлдын элементинин суммасы', Summa);
 Close(f);
 end.
```

12.4. Файлдар менен иштөө үчүн жалпы каражаттар

12.4.1. System модулунун процедуралары жана функциялары

System модулунун процедура жана функцияларын чакырууга ар дайым уруксат жана кошумча модулдарды кошууну талап кылбайт. Жогоруда каралган процедуралардан жана функциялардан сырткары бул модуль файлдар менен иштөө үчүн файлдарды кайра атоо, өчүрүү процедураларын жана каталогдор менен иштөө процедураларын кармап турат.

<u>Каталогдор менен иштөө процедуралары</u>

Каталогдор менен иштөө үчүн system модулуна мааниси боюнча MS-DOS тун командаларына окшош ChDis, MkDir, RmDir жана GetDir процедуралары кошулган. Бул процедуралардын колдонулушунун мисалы катары төмөнкү программаны келтиребиз.

```
Program  Directories_Demo;
    Uses  Crt;
    Var     S:String;
    Begin  ClrSer;
      {Учурдагы каталог деп E дискинин түпкү катологун алуу}
   ChDir (′E:/′);
            {Учурдагы дискти жана каталогду көрсөтүү}
   GetDir(0,S);
  Writeln(′Учурдагы диск жана каталог:’, S);
  {MyDir камтылуучу каталогун түзүү}
  MkDir(′MyDir′);
  {MyDir камтылуучу каталогуна өтүү}
  ChDir (′MyDir′);
  { учурдагы диск жана каталогду көрсөтүү}
  GetDir(0,S);
  Writeln(′Учурдагы диск жана каталог:′,S);
 {учурдагы каталог катары E дискинин түпкү каталогун алуу}
  ChDir(′ \ ′);
  {MyDir камтылуучу катологун өчүрүү}
  RmDir (′MyDir′);
  { учурдагы диск жана катологду көрсөтүү}
  GctDir(0,S);
  Writeln(′Учурдагы диск жана католог :′,S);
End.
```

Файлдарды кайра атоо жана өчүрүү процедуралары

Rename процедурасы физикалык файлдарды кайра атоо (атын өзгөртүү) үчүн, Erase процедурасы аларды өчүрүү үчүн кызмат кылат. Бул процедураларды кандайдыр бир физикалык файл менен байланыштырылган, бирок али алар үчүн файлды ачуу амалы аткарыла элек файлдын өзгөрүлмөлөрү үчүн гана аткарууга болоорун байкоого болот. Төмөнкү программа адегенде MyFile.Dat файлын Result.Dat файлына кайра атоону, андан кийин бул файлды өчүрүүнү демонстрациялайт.

```
Program RemDelDemo;
   Var
        F:file;
  Begin
   Assing(f,`MyFile:Dat`);
   Rename(f,`Result.Dat`);
  { Кайра аталган файл менен болгон аракеттер }
   Erase(f);
  End.
```

12.4.2. Dos модулунун константалары, типтери, процедуралары жана функциялары

Dos модулу тарабынан ишке ашырылуучу файлдын каражаттары пайдаланылган программалар Dos идентификаторун көрсөтүү менен жазылган uses сүйлөмүн кармап турушу керек.

```
Program Example;
  Uses Dos;
            . . .
  begin
            . . .
  end.
```

Файлдар менен иштөө үчүн типтер

```
TFileRec=record
            Handle: Word;
            Mode: Word;
            RecSize: Word;
            Private : array[1..26] of Byte;
            UserData: array[1..16] of Byte;
            Name: array[o..79] of Char;
          end;

 PTextBuf = ^TTextBuf;
 TTextBuf = array[0..127] of char;
 TTextRec = record
    Handle:Word;
    Mode: Word;
    BufSize:Word;
   Private: Word;
   BufPos:Word;
   BufEnd: Word;
   BufPtr: PTextBuf;
   OpenFunc: Pointer;
    InOutFunc:  Pointer;
    FlushFunc: Pointer;
    CloseFunc: Pointer;
    UserData:   array[1..16] of Byte;
    Name: array [0..79] of Char;
    Buffer: TTextBuf;
 end;
```

TFileRec тиби типтештирилген да, типтештирилбеген да файлдар үчүн, ал эми TTextRec тиби тексттик файлдар үчүн гана пайдаланылат.

```
SearchRec=record
    Fill: array[1..21] of Byte;
   Attr: Byte;
    Time: Longint;
    Size:  Longint;
    Name:string[12]
end;
```

SearchRec тибиндеги өзгөрүлмөлөр FindFirst жана FindNext процедураларында файлдардын катологдорун көрүү үчүн пайдаланылат.

```
DateTime=record
  Year, Month, Day, Hour,
   Min, Sec, Word;
   end;
```

DateTime тиби UnpackTime жана PackTime процедураларында күндү жана убакытты кармап турган төрт байттык маанини анализдөө, таңгактоо (упаковать) жана түзүү үчүн пайдаланылат. Кийин бул төрт байттык маани GetFTime, SetTime, FindFirst жана FindNext процедураларында пайдаланылат.

Dos модулунун ComStr, PathStr, DirStr, NameStr, ExtStr жолчолук типтери Fsplit жолчолук процедурасын чакырууда файлдардын аттары жана маршруттар менен иштөө үчүн пайдаланылат.

Файлдар менен иштөө үчүн константалар

```
fmClosed=$0780;
fmInput=$0781;
fmOutput=$0782;
fmInOut=$0783.
```

fmClosed, fmInput, fmOutput, fmInOut константалары файлдарды ачууда жана жаабууда TTextRec жана TFileRec тибиндеги өзгөрүлмөлөр Mode (окуу-жазуу режими) талаасынын маанилерин коюу үчүн пайдаланылат.

```
ReadOnly =$01      {окуу үчүн гана       }
Hidden   =$02      {«Бекитилген» файл}
```

```
Sysfile     = $04       {системалык файл   }
VolumeID  = $08         {томдун аты          }
Directory =  $10        {каталог               }
Archive    = $20        {архивдик файл       }
Anyfile     = $3F       {каалагандай файл  }
```

ReadOnlu, Hidden, SysFile, VolumeID, Directory, Archive, AnyFile константалары файлдын атрибуттарын коюу үчүн пайдаланылат.

<u>Файлдар менен иштөө үчүн процедуралар жана функциялар</u>

GetFTime	файлдын акыркы жазылыш күнүн жана убактысын берет.
SetFTime	файдын акыркы жазылыш күнүн жана убактысын коет.
PackTime	DateTime жазуусун SetFTime процедурасы тарабынан пайдаланулуучу Longint тибиндеги күндү жана убакытты көрсөтүүнүн ички кодуна өзгөртүп түзөт.
UnpackTime	GetFTime, FindFirst, FindNext процедуралары тарабынан кайтарылып берилүүчү Longint тибиндеги күндүн жана убакыттын көрсөтүлүшүнүн ички DateTime жайылтылган (распакованный) жазуусуна өзгөртүп түзөт.
FExpand	файылдын атын кабыл алат жана толук такталган атын (дискти, катологду, кеңейтирилишин) берет
FSearch	каталогдордун тизмесинде файылды издейт.
FindFirst	берилген (же учурдагы) катологдо мазмуну файлдын берилген аты жана атрибуттары менен дал келген жазууну издөөнү жүргүзөт.
FindNext	файлдын аты жана атрибуттар FindFirst процедурасына мурунку кайрылууда берилген берилгендер менен дал келүүчү кийинки жазууну берет.
GetFAttr	файлдын атрибуттарын берет.
SetFAttr	файлдын атрибуттарын коет

Мындан сырткары файлдар менен иштөөдө (өзгөчө жаңы файлдарды түзүүдө) Dos тун дагы эки функциясы пайдалуу.

DiskFree – берилген дисксалгычта (дискаводдо) турган дисктеги бош байттардын санын берет.

DiskSize - берилген дисктин байттардагы толук көлөмүн берет.

12.5. Типтештирилген файлдар

Типтештирилген файлдын бардык элементтери бир типте болушу керек. Типтештирилген файлдар файлдык жана ага таянуучу типтерден башка каалаган типте боло бериши мүмкүн.

Туура эмес баяндоонун мисалы:

```
Type TF1= file of file;
   TFR = record
    A: Integer;
    F: file of real;
end;
TF2: file of  TFR;
```

Типтештирилген файлдар менен иштөөдө удаалаш да түз да кайрылууга уруксат берилет. Түз кайрылуу менен иштеп жатканда типтештирилген файлдардын элементтери ар дайым нөлдөн баштап номерленээрин эстен чыгарбоо керек.

12.5.1. Типтештирилген файлдар менен иштөө үчүн процедуралар жана функциялар

*Типтештирилген файлдардан окуу – дайым **Read** процедурасы менен гана, ал эми жазуу - **Write** процедурасы менен гана аткарылат. Бул учурда файлдын тиби менен бирдей типте болгон өзгөрүлмө гана окуунун/жазуунун бирдиги боло алат.*

Read процедурасы типтештирилген файлдар үчүн төмөнкүдөй форматка ээ:

```
Read(файлдык өзгөрүлмөнүн аты,өзгөрүлмөлөрдүн тизмеси)
```

Read процедурасынын тизмесиндеги ар бир өзгөрүлмөгө (в переменную) окуган учурда, файлдагы учурдагы позициянын көрсөткүчү кийинки элементке которулат. Эгерде файлдын учурдагы позициясынын көрсөткүчү акыркы элементте турган болсо, б.а. файлдын аягында (Fof(f)=True), анда Read процедурасынын аткарылышы аткаруу убактысынын катасына алып келет.

Write процедурасы типтештирилген файлдар үчүн төмөндөгүдөй форматка ээ:

```
Write(файлдык өзгөрүлмөнүн аты, өзгөрүлмөлөрдүн тизмеси)
```

Файлга жазууну аткарууда эсте тутуучу нерсе, ар бир өзгөрүлмөгө жазганда файлдагы учурдагы позициянын көрсөткүчү, окуган учурдагыдай, кийинки элементке жылат. Эгерде учурдагы позициянын көрсөткүчү акыркы элементте турса, б.а. файлдын аягында (Eof(f)=True) турса, анда Write процедурасын аткарганда файл кеңейет.

Түз кайрылуудагы файлдар менен иштөө үчүн төмөнкү процедуралар жана файлдар арналган.

FilePos	файлдагы көрсөткүчтүн учурдагы позициясынын номерин берет (позициялар нөлдөн баштап номерленишет)
FileSize	файлдын учурдагы өлчөмүн берет (бирден баштап эсептегендеги файлдын элементтеринин саны).
Seek	файлдагы учурдагы позициянын көрсөткүчүн берилген номердеги элементке жылдырат(нөлдөн баштап эсептегенде).
Truncate	файлдын өлчөмүн учурдагы позицияга чейин кесет. Учурдагы позициядан кийин турган файлдагы баардык элементтер өчүрүлөт да файлдагы учурдагы позиция анын аягы болуп калат (Eof(f)=True).

Типтештирилген файлдын элементтерине түз кайрылуу менен иштөөгө мисал катары алмашуу (“көбүкчө”) усулу менен иштөөчү файлды сорттоонун программасын карайбыз.

```
Preogram FileSort;
   Uses crt;
    Var
       F:  file of Integer;
      X,Y: Integer;
      i, j:  Longint;
Begin
   {$I-}
  Assign (f, `Sort. Dat`);
  Reset(f)
   {$I+}
if IOResult <> 0 then
begin
   Writeln (`Файлды ачуу катасы`);
Halt(1);
end;
ClrScr;
Writeln (`Баштапкы файл:`);
for i:=1  to FileSize (f) do
```

```
  Begin
      Read(f, x);
      Write(x: 8);
end;

Writeln;
    Close(f);
{- - - - - - - - - - - - - - - - - - - - - - - - - - - -}
Reset(f);
For i:= FileSize(f)-1 downto 1 do
{ кезектеги максималдык элементтин }
{i-позицияга калкып чыгышы}
  for j:=0 to i-1 do
 begin
    Seek(i, j);
     Read(f, X, Y);
If  X > Y then
 Begin
     Seek(f, j);
     Write (f, Y, X);
        end;
end;
  Close(f);
{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
Reset(f);
 Writeln('Сорттолгон файл:');
   For i:=1 to FileSize (f) do
 Begin
   Read(f, X);
   Write(X:8)
end;
Close (f);
   end.
```

Типтештирилген файлдын тиби – бул физикалык файлга (файлдын үстүнө) анын элементтерине кайрылуу үчүн биз «коё турган шаблон» («терезе») экендигин эске салабыз. Эгер андай боло турган болсо, суроо пайда болот: «Бир шаблонду» пайдаланып файлды түзүп, бирок аны башка «шаблонду» пайдаланып окууга болобу?. Чындыгында эле ушундай кылууга болот. Мисалы, төмөнкү программада, адегенде физикалык файл Byte тибиндеги файл катары ачылат, натыйжада андагы жазылган символдордун ASCII коддору печатка чыгат.

```
Program CharToByte;
 uses Crt;
var
   FC :  file of Char;
   FB :  file of Byte;
   Ch :  Char;
   B   :  Byte;
Begin ClrScr;
  Assign (FC ,'Test.Dat');
  Rewrite(FC);
  For  Ch := `0`  to `9`  do  write (FC, Ch);
  For  Ch := `A`  to `J`  do  write (FC, Ch);
  Close (FC);

Assign (FB, `Test.Dat`);
Reset (FB);
While not Eof(FB) do
Begin
      Read(FB, B);
      Write(B: 8);
end;
Close(FB)
end.
```

Жыйынтыгы:

```
  48   49   50   51   52    53   54   55   56   57
  65   66   67   68   69    70   71   72   73   74
```

12.6. Тексттик файлдар

12.6.1. Тексттик файлдын структурасы

Тексттик файлдар Turbo Pascal тилиндеги file of Char тибиндеги файлдардын түрүнө кирүүчү өзгөчө түрдөгү файлдар болуп эсептелет.

Тексттик файлдарды баяндоо үчүн Text алдын ала аныкталган тиби пайдаланылат.

```
Var
  TextFile: Text;
```

Тексттик файлдарда файлдын аяктоо белгиси Eof менен бирге дагы жолчонун аяктоо белгиси – Eoln пайдаланылат. Eoln белгиси – бул эки символдун – ASCII коду 13 болгон (“Каретканы кайтаруу”) жана 10 болгон (“жолчону которуу”) символдордун удаалаштыгы болуп эсептелет.

Тексттик файлды образдуу түрдө ар бир жолчосунун аягында Eoln белгиси турган китептин бир бети катары элестетүүгө болот.

Символдун коду	Символ дун коду	…	Eoln		
Символдун коду	Символдун коду	…	Символдун коду	Символдун коду	Eoln
Символдун коду	Символдун коду	…	Символдун коду	Eoln	
Символдун коду	Символдун коду	…	Eoln		
Символдун коду	Символдун коду	…	Символдун коду	Символдун коду	Eoln

Кийрүү-чыгаруунун стандарттык файлдары Input жана Output тексттик файлдар экенин эстейли. Read, Readln, Write, Writeln процедураларын стандарттык кийрүү-чыгарууда пайдалануну биз 9-бапта карап көргөн элек, тексттик файлдар үчүн дагы так ошондой эле болот, айырмасы бул процедуралардын биринчи параметри болуп файлдык өзгөрүлмө көрсөтүлүшү керек.

```
Read (f, A, B);
  Write (g, 'A=', A, 'B=', B);
  Readln(f, C, D);
  Writeln(g, 'C=', C, 'D=', D);
```

12.6.2. Текстик файлдын file of Char дан айырмачылыктары

1. Маанилерди файлга жазууда сандык берилгендердин, символдордун чынжырчасына автоматтык түрдө өзгөртүлүп түзүлүшү, жана тиешелүү сандык типтерди өзгөрүлмөлөргө (в переменные) окуу амалынын аткарылышында сандар болуп эсептелген символдордун кайрадан сандык маанилерге өзгөртүлүп түзүлүшү.

2. Текстик файлдар түз кайрылууга ээ эмес.

3. Текстик файлга жазуу жана текстик файлдан окуу үчүн кээ бир стандарттык типтеги гана өзгөрүлмөлөр уруксат берилет.

4. Жолчонун аяктоо белгисинин болушу.

5. Текистик файлдан окуу жана ага жазуу үчүн башка типтеги файлдарга уруксат берилбеген Readln жана Writeln процедураларын пайдалануга уруксат берилген.

12.6.3. Тексттик файлдар менен иштөө үчүн процедуралар жана функциялар

Тексттик файлдар үчүн жалпы процедура-функцияларга кошумча төмөнкү процедураларды жана функцияларды пайдаланууга болот.

Append	акырына (аягына) жаңы элементтерди кошуу үчүн бар болгон (существующий) файлды ачат.
Flush	тексттик файл үчүн чыгаруу буферин таштайт (сбрасывает).
Readln	Read сыяктуу эле иштейт, бирок кошумча учурдагы жолчонун калган бардык элементтерин өткөрүп жиберүүнү аткарат жана файлдын учурдагы позициясынын көрсөткүчүн файлдын кийинки жолчосунун башталышына жылдырат.
SeekEof	тексттик файл үчүн Eof (файлдын бүтүшү) абалын берет
SeekEoln	тексттик файл үчүн Eoln (жолчонун бүтүшү) абалын берет.
SetTextBuf	тексттик файл үчүн кийирүү-чыгаруу буферин дайындайт.
Writeln	Write сыяктуу эле иштейт, бирок процедурада көрсөтүлгөн маанини жазуудан кийин, кошумча тексттик файлга (в файл) Eoln - жолчонун бүтүү белгисин жазат.

12.7. Типтештирилбеген файлдар

Типтештирилбеген файлды баяндоодо file кызматчы сөзү гана көрсөтүлөт, мисалы,

```
Var F: file;
```

Типтештирилбеген файлдык өзгөрүлмөлөр файлдар менен болгон төмөнкү деңгээлдеги иштер үчүн арналган.

Алардын жырдамында каалаган типтеги жана логикалык структурадагы файлга, символдук файлга Byte тибиндеги файлдык өзгөрүлмөнүн жардамында кайрылуунун аткарылышына окшош кайрылууга болот. Айырмасы, типтештирилбеген файл типтештирилген файлга окшоп окуу/жазуунун так коюлган бирдигине ээ эмес. Типтештирилбеген файлдарда бир жолку кайрылууда буфердин чоңдугуна барабар болгон байттардын саны окулат/жазылат, ал болсо файлдар менен иштөө тездигин жогорулатат. Типтештирилбеген файлдардын кийирүү/чыгаруу буфери катары каалагандай өзгөрүлмө келе алат.

12.7.1. Типтештирилбеген файлдар менен иштөө үчүн процедуралар жана функциялар

Типтештирилбеген файлдар менен иштөөдө типтештирилген файлдарга колдонууга мүмкүн болгон процедуралардын жана функциялардын дээрлик бардыгын колдонууга болот. Айырмалап кароого туура келген жери Read жана Write процедураларынын ордуна BlockRead жана BlockWrite процедуралары пайдаланылат, ал эми Reset жана Rewrite процедуралары берилгендерди берүүдө пайдаланылуучу жазуунун өлчөмүн аныктоочу Word тибиндеги экинчи параметрге ээ боло алышат. Эгер бул параметр жок болсо, анда көрсөтүлбөгөн учурда (по умолчанию) жазуунун өлчөмү 128 байтка барабар деп кабыл алынат.

1 – ТИРКЕМЕ. КАТАЛАР

Жаңылбас жаак,
мүдүрүлбөс туяк болбойт.
(Эл макалы)

Биз баарыбыз өзүбүздүн турмуштук тажрыйбабыздан көрүп жүргөндөй каталар кадам сайын кезигет. Алар ар төрдүү болот: критикалык каталар болот - ситуцияны оңдош үчүн токтоосуз бир нерселерди жасоо зарылдыгын талап кылган каталар, эскертүү иретиндеги каталар болот - ага жооп кылып жоюп койсоң азаматсың, жооп кылбый койсоң - учуру келгенче эч кандай чара колдонбой койсо болчудай, а балким билинбей өтүп кетээр …

Программаларды талдоодо деле так мына ушундай – себеби программа деле адам ишмердигинин бир жыйынтыгы. Программаларда каталар, кадимки турмуштагыдай эле, дароо болбосо да, ар кандай эле пайдалануучу үчүн болбосо да жетишээрлик жакшы көрүнөт. Пайдалануучулар каталарды байкаар замат же программаны талдап-иштеп чыккандарды шыбай башташат, же аны жакшыраак башка программа менен алмаштырууну көздөшөт, же эгер убактысы жана каалоо болгон болсо ушундай калтыс жерлерди айланып өтүүгө аракет жасашат – канткен менен иштеш керек да.

Ошон үчүн программаны жазууга чейин жана жазып жатканда бул программа эмне кылышы керек, каталар кетирилип калчу кандай ситуациялар болушу мүмкүн экендигин терең ойлонуштуруп койгон өтө чоң мааниге ээ. А бирок реалдуу турмушта тескерсинче - оболу программаны жазып анан кийин ойлонобуз.

Профессионалдар эмне деп ойлошот, окуп көрөлү. **Чоң программалык система көп жылдык тестирлөөдөн жана пайдалануудан кийин деле эч качан аягына чейин оңдолуп-түзөлүп бүтпөйт**. *Аягына чейин кичинекей программалар гана оңдолуп-түзөлүшү мүмкүн, андан ары чоңураак жана татаалыраак кылганга болбойт. Программанын жалпы алгоритми үчүн өтө эле көп жолдорду караштырса болот, киргизилүүчү берилгендердин же пайдалануучулардын аракеттеринин вариантары да өтө эле көп. Жүздөгөн жылдарга созулган тестирлөө да, - эгер, албетте ушундай кылуу мүмкүн болсо - чоң, татаал программанын мүмкүн болгон бардык бутактарын текшерип чыгууга жетмек эмес. А бирок ага карабастан, «каталардан арылган программалык камсыздандыруу» жөнүндө асыресе сөз кылган адамдар табылат*, - Джозеф Фокс, Демек мындан, эмне үчүн Microsoft Windows, Microsoft Visual Studio сыяктуу чоң программалык системалар ар убак жаңыланып чыгып тургандыгы түшүнүктүү. Бирок прогресс ордунда турган жок:

программистерге жардамга алардын өздөрү тарабынан шаблондор (Templates), программаларды долбоорлоо жана талдоонун визуалдык каржаттары, программалоонун жаңы технологиялары иштелип чыгылып жатат.

Кантсе да негизги каталарды алдын ала көрө билсе болот, ал үчүн алар кандай болоорун билүү жана али программанын жазылыш стадиясында андай каталарга жол койбоо керек. Жакшысы, даро эле катасыз, жакшы программаларды жазууга көнүү керек – бул жалпыбыз үчүн акырында жеңил болот (арзанга түшөт).

Программалардагы каталардын категориялары

Белгилүү программисттер Крис Х. Паппас жана Уильям Х. Мюррей тарабынан сунуш кылынган каталардын классификациясын пайдаланабыз.

- **Синтаксистик каталар.** Мындай каталар компиляция этабында программисттин программалоо тилинин кандайдыр бир конструкциясын билбегендигинен же жөн эле кош көңүлдүгүнөн улам пайда болот. Мындай каталар менен компиляторлор өздөрү жакшы иш алып бара алат. Ошон үчүн мындай каталар бир аз дүүлүгүүнү пйда кылганы менен андай эле коркунучтуу деп эсептелинбейт. Андан сырткары компиляторлор *эскертүүлөрдү* чыгарып бере алат, аларды көптөгөн тажрыйбалуу программисттер логикалык каталардын потенциалдуу коркунучу катары кароого ыкташат. Өтө өркүндөтүлгөн компиляторлор болот, алар өздөрүнүн эскертүүлөрүндө өтө сейрек адашат (мисалы, Microsoft Visual C++ 6.0, Borland C++ 5.02).
- **Компоновщиктин каталары.** Мындай каталар көбүнчө ар түрдүү модулдарды туташтыруу (стыковка) максатында программист тарабынан интерфейстин тура эмес уюштурулганынан улам пайда болот. А эгерде система тилдердин бирөөсүндө модулдар менен иштеп жаткан болсо мындай каталардын себептеринин эң ыктымалдуусу – библиотекалардын жана башка кошулуучу файлдардын директорияларынын туура эмес көрсөтүлүшү болот.
- **Аткаруу убактысынын каталары.** Мындай каталар программанын мөөнөтүнөн мурда авариялык токтошуна же циклденип калышына алып барат:
  - *аппараттык аныкталуучу каталар:* «Нөлгө бөлүү», «Нөлдүн же терс сандын логарифмин эсептөөгө болгон аракет», эстин корголушунун бузулушу, түзүлүштөрдүн каталары ж.б.
  - *системалык каталар:* туура болбой калган (неудачная) файлдык амал, билдирүүлөр кезегинин толуп ашып кетиши.

- *нагыз программалык каталар:* индекстин массивдин чегинен чыгып кетиши, бош кезектен элементти жоготуу (алып салуу), идея боюнча эч качан пайда болбошу керек болгон чечимдин бутагына чыгуу ж.б.
- *тиркемелер үчүн спецификалык болуп эсептелген каталар:* мисалы, чыгаруунун тура эмес форматы.

♦ **Логикалык каталар**. Бул каталардын эң татаал түрү, анткени алар дароо эле байкала койбойт, аларды алдын ала көрө билүү түздөн-түз программисттин квалификациясынан, маданиятынан жана ошондой эле буйрутмачы тарабынан маселенин тура коюлушунан көз каранды. Так мына ушундай каталарды издеп табуу жана локалдаштыруу үчүн оңдоп-түзөө (отладка) процесси жашайт.

Төмөндө Turbo Pascal тилинин чөйрөсүндө алынышы мүмкүн болгон каталардын тизмегин алардын түшүндүрмөлөрү менен келтиребиз.

Каталар жөнүндө билдирүүлөр

Компилятордун билдирүүлөрү

1.	Out of memory – Эстин чегинен сырткары чыгып кетүү
2.	Identifier expected – Идентификатор көрсөтүлгөн эмес. Бул жерде идентификатор турушу керек.
3.	Unknown identifier – Белгисиз идентификатор. Идентификатор баяндалган эмес.
4.	Duplicate identifier – Кайталанган идентификатор. Бул идентификатор учурдагы блокто эбак баяндалган программанын, костантанын, модулдун, өзгөрүлмөнүн, типтин, процедуранын же функциянын атын көрсөтөт.
5.	Syntax error – Синтаксистик ката. Баштапкы текстте тура эмес символ табылды.
6.	Error in real constant – Чыныгы константада ката бар.
7.	Error in integer constant – Бүтүн константада ката бар.
8.	String constant exceeds line – Жолчолук константа жолчонун өлчөмүнөн ашып кетти.
9.	Too many nested files – Камтылма файлдар өтө эле көбөйүп кетти.
10.	Unexpected end of file – Файлдын тура эмес бүтүшү.
11.	Line to long – Жолчо өтө эле узун. Жолчонун узундугу 126 символдон ашып кетти.
12.	Type identifier expected – Типтин идентификатору керек. Бул жерде идентификатордун тиби турушу керек.
13.	Too many open files – Ачылган файлдар өтө көп.

14.	Invalid file name – Файлдын аты туура эмес. Файлдын аты туура эмес же жок эле жолду көрсөтүп турат.
15.	File not found – Файл табылган жок.
16.	Disk full – Диск толуп калды.
17.	Invalid compiler directive – Компилятордун туура эмес директивасы.
18.	Too many files – Файлдардын саны өтө эле көп. Программанын же программалык модулдун компиляциясында катышкан файлдардын саны өтө эле көп.
19.	Undefined type in pointer definition – Көрсөткүчтү аныктоодо аныкталбаган тип бар. Баяндалбаган көрсөткүчтүк типке шилтеме (ссылка) иштетилген.
20.	Variable identifier expected – Өзгөрүлмөнүн идентификатору керек.
21.	Error in type – Типти аныктоодогу ката. Типтин аныктамасы мындай символ менен баштала албайт.
22.	Structure too large – Өтө эле чоң структура. Структуралык тип үчүн максималдык уруксат берилген өлчөм 65535 байт.
23.	Set base type out of range – Көптүктүн базалык тиби үчүн чек-ара бузулган. Көптүктүн базалык тиби – 0 дөн 255 ке чейин болгон кесинди жа маанилеринин саны 256 дан ашпаган саналуучу тип болушу керек.
24.	File components may not be files or objects – Файлдын компоненттери файлдар же обьекттер боло албайт.
25.	Invalid string Length – Жолчонун узундугу тура эмес. Жолчонун баяндалуучу максималдык узундугу 1 ден 255 ке чейинки аралыкта болушу керек.
26.	Type mismatch –Типтердин тиешелеш келбестиги.
	Invalid subrange base type – Тип кесиндисинин туура эмес базалык тиби. Бардык иреттик типтер уруксат берилген базанын типке ээ болушу керек.
28.	Lower bound greater than upper bound – Төмөнкү чек жогорку чектен чоң.
29.	Ordinal type expected – Иреттик тип керек.Чыныгы, жолчолук, структуралык жана иреттик типтерге, бул учурда уруксат берилбейт.
30.	Integer constant expected – Бүтүн копетанта керек
31.	Constant expected – Константа керек.
32.	Integer or real constant expected – Бүтүн же чыныгы константа керек.
33.	Pointer type identifier expected – Көрсөткүчтүк типтеги индентификатор керек. Бул индентификатор көрсөткүчтүк типти бере албайт.

34.	Invalid function result type – Функиянын жыйынтыгынын тиби туура эмес. Функциянын жыйынтыгынын туура тиби болуп бардык жөнөкөй, жолчолук жана шилтемелик жолчолук типтер эсептелет.
35.	Label indentifier expected – Эн-белгинин (метканын) идентификатору керек. Эн-белги идентификатор менен белгиленген эмес.
36.	BEGIN expected – BEGIN болушу керек.
37.	END expected – END болушу керек.
38.	Integer expression expected – Integer тибиндеги туюнтма керек.
39	Ordinal expression expected – Иреттик типтеги туюнтма керек. Туюнтма иреттик типке ээ болушу керек.
40.	Boolean expression expected – Boolean тибиндеги туюнтма керек. Туюнтма Boolean тибине ээ болушу керек.
41.	Operand types do not match operator – Операндардын типтери операторго тура келбейт.
42.	Error in expression – Туюнтмада ката бар.
43.	Illegal assignment – Ыйгаруу тура эмес. Типтештирилбеген файлдарга жана өзгөрүлмөлөргө маанилерди ыйгарууга болбойт.
44.	Filed identifier expected – Талаа идентификатору керек. Бул идентификатор мурда келген (предшествующей) жазуу тибиндеги өзгөрүлмө үчүн талааны бере албайт.
45.	Object file too large – Обьекттик файл өтө элө чоң. Turbo Pascal өлчөмү 64К дан ашкан .OBJ – файлды компановка кыла албайт.
46.	Undefined external – Тышкы процедура аныкталган эмес. Обьекттик файлда тышкы процедура же функция тиешелүү Public аныктамасына ээ эмес.
47.	Invalid object file record – Обьекттик файл тура эмес жазылган. .OBJ – файл тура эмес обьекттик жазууну кармап турат.
48.	Code segment too large – Код (командалар) сегменти өтө эле чоң. Программанын же программалык модулдун максималдык өлчөмү 65520 байтка барабар.
49.	Data segment too large – Берилгендер сегменти өтө эле чоң. Программада берилгендер сегментинин максималдык өлчөмү пайдаланылуучу программалык модулдарда баяндалган берилгендер менен кошо эсептегенде 65520 байтка барабар.
50.	Do expected – Do кызматчы сөзү керек.
51.	Invalid PUBLIC definition – PUBLIC туура эмес аныкталган.
52.	Invalid EXTPN definition – EXTPN туура эмес аныктылган.
53.	Too many EXTPN definition – EXTPN аныктамалары өтө көп. Tyrbo Pascal 256дан көп EXTPN аныктамалары болгон учурда OBJ-файлды иштете албайт.
54	OF expected – OF талап кылынат.
55.	INTERFACE expected –Интерфейстик секция талап кылынат.

56.	Invalid relocatable reference – уруксат берилбеген аралашма шилтеме.
57	THEN expected – THEN талап кылынат. THEN кызматчы сөзү жок болуп жатат.
58.	TO or DOWNTO expected – TO же DOWNTO талап кылынат. TO же DOWNTO кызматчы сөзү жок болуп жатат.
59.	Undefined forward – Мурда келүүчү (опережающее) баяндама аныкталган эмес.
61.	Invalid typecast – Типтин өзгөртүп түзүлүшү туура эмес.
62	Division by zero – Нөлгө бөлүү болуп жатат.
63.	Invalid file type – Туура эмес файлдык тип. Бул файлдык тип файлдарды иштетүү процедурасы тарабынан тейленбейт.
64.	Cannot Read or Write variables of this type – Мындай типтеги өзгөрүлмөлөрдү окууга же жазууга болбойт.
65.	Pointer variable expected – Өзгөрүлмө-көрсөткүчтү пайдалануу керек.
66.	String variable expected – Жолчолук өзгөрүлмө керек.
67.	String expression expected – Жолчолук типтеги туюнтма керек.
68.	Circular unit reference – Модулга болгон циклдик шилтеме бар. Interface секциясында эки модул бири-бирине шилтеме жасай албайт.
69.	Unit neme mismatch – Программалык модулдардын аттарынын тиешелеш келбестигин .TPU, .TPW же .TPP файлында табылган программалык модулдун аты uses операторунда көрсөтүлгөн атка туура келет.
70.	Unit version mismatch – Программалык модулдардын версияларынын туура келбестиги. Бул программада пайдалануучу бир же бир нече программалык модулдар алардын компиляциясынан иштеп өзгөргөн.
71.	Internal stack overflow – Ички стектин толуп-ашышы. Операторлордун камтылуучулугунун өтө эле чоң деңгээлинен улам компилятордун ички стеги толук иштетилип бүткөн.
72.	Unit file format error – Программалык модулдун файлынын форматы ката. .TPU, .TPW же .TPP файлы (платформадан көз каранды түрдө) жараксыз.
73.	Implementation expected – Ишке ашыруу (реализация) секциясы керек. Implementation кызматчы сөзү жок.
74.	Constant and case types do not match – Константалардын типтери жана case операторунун туюнтмасынын тиби бири-бирине тура келбейт.
75.	Record variable expected – Жазуу тибиндеги өзгөрүлмө керек.
76.	Constant out of range – Константа чек араны бузуп жатат.
77.	File variable expected – Файлдык өзгөрүлмө керек.

78.	Pointer expression expected – Көрсөткүч тибиндеги туюнтма керек, туюнтма көрсөткүчтүк типке ээ болушу керек.
79.	Integer or real expression expected – Real же Integer тибиндеги туюнтма керек. Туюнтма Real же Integer тибине ээ болушу керек.
80.	Label not within current block – Эн-белги (метка) учурдагы блоктун ичинде жайгашкан эмес. Goto оператору учурдагы блоктон сырткары жаткан эн-белгиге шилтеме жасай албайт.
81.	Label already defined – Эн-белги эбак аныкталган. Бул эн-белги менен оператор белгиленген.
82.	Undefined label in processing statement part – Операторлордун иштетилип жаткан бөлүгүндө аныкталбаган эн-белги (метка) бар.
83.	Invalid @ argument – @ операторунун аргументи жараксыз. Жарактуу аргумент болуп өзгөрүлмөлөргө болгон шилтемелер жана процедуралардын же функциялардын идентификаторлору эсептелет.
84.	UNIT expected – UNIT кызматчы сөзү керек.
85.	«;» expected – «;» керек.
86.	«:» expected – «:» керек.
87.	«,» expected – «,» керек.
88.	«(» expected – «(» керек.
89.	«)» expected – «)» керек.
90.	«=» expected – «=» керек.
91.	«:=» expected – «:=» керек.
92.	«[» or «(.» expected – «[» же «(.» керек.
93.	«]» or «.)» expected – «]» же «.)» керек.
94.	«.» expected – «.» керек.
95.	«..» expected – «..» керек.
96.	Too nany variables – Өзгөрүлмөлөр өтө эле көп.
97.	Invalid FOR control variable – FOR операторунун башкаруучу өзгөрүлмөсү тура эмес. FOR опреаторунун башкаруучу өзгөрүлмөсү учурдагы камтылуучу программанын баяндоолор бөлүгүндө аныкталган саналуучу типтеги өзгөрүлмө болушу керек.
98.	Integer variable expected – Бүтүн типтеги өзгөрүлмө керек.
99.	Files are not allowed here – Бул жерде файлдарга уруксат жок. Типтештирилген константа файлдык типке ээ боло албайт.
100.	String length mismatch – Узундктун тиешелеш келбестиги. Жолчолук константанын узундугу символдук массивдин элементтеринин санына тура келбейт.
101.	Invalid ordering of fields – Талаалардын ирети тура эмес. Жазуу тибиндеги константада талаалар алардын баяндалыш тартибинде жазылышы керек.
102.	Sting constant expected – Жолчолук типтеги константада.
103.	Ordinal or real variable expected – Integer же Real тибиндеги өзгөрүлмө керек.

104.	Ordinal variable expected – Иреттик типтеги өзгөрүлмө керек.
105.	INLINE error – INLINE операторунда ката бар. < операторун өзгөрүлмөлөргө болгон аралашма (перемещаемый) шилтемелер менен айкалыштырууга уруксат жок.Мындай шилтемелер дайыма сөз өлчөмүнө ээ.
106.	Character expression expected – Мындан мурда келүүчү туюнтма символдук типке ээ болушу керек.
107.	Too many relocation items – Аралашма элементтер өтө эле көп (Реалдык режим үчүн гана) .EXE файылдын которулуш табилицасы бөлүгүнүн өлчөмү 64K тан ашып кеткен.
108.	Overflow in arithmetic operation – Арфиметикалык амалда толуп-ашуу (переполнение) бар, амалдын жыйынтыгы Longint(-2147483648 .. 2147483647) аралыгында жатпай калды.
109.	No enclosing FOR, WHILE or REPEAT statement – Иштетүүчү FOR, WHILE же REPEAT оператору жок. Break жана Continue стандарттык процедуралары for, while же repeat операторлорунан сырткары пайдаланыла албайт.
112.	CASE constant out of range – Case тин константасы диапазондон сырткары жатат.Бүтүн сандык Case операторлору үчүн константалар -32768 дан 32767ге чейинки аралыкта жатышы керек.
113.	Error in statemend – Операторду ката бар. Бул индентифи- катор операторду баштай албайт.
114.	Cannot call an interrupt procedure – Үзгүлтүк процедурасын чакырууга мүмкүнчүлүк жок. Үзгүлтүк процедурасын түздөн түз чакыруу мүмкүн эмес.
116.	Must be in 8087 mode tp compile this – Компиляция үчүн 8087 режими зарыл. Бул конуструкция {$N+} режими учурунда гана копиляциялана алат.
117.	Target address not found – Баруу адреси (Адрес назначения) табылган жок.
118.	Include files are not allowed here – Мындай ситуацияда кошулуучу файлдарга уруксат берилбейт. Ар бир операторлор бөлүгү бүтүн бойдон бир файлдын ичинде жайгашышы керек.
119.	No inherited methods are accessible here – Бул жерде мурасталуучу усулдарга уруксат жок. **I**nherited кызматчы сөзү усулдан сырткары же текке ээ болбогон усулда же объекттик типте пайдаланылып жатат.
121.	Invalid gualifier – Квалификатор туура эмес.
122.	Invalid variable reference – Өзгөрүлмөгө болгон уруксат берилбеген шилтеме. Мурда келген конструкция өзгөрүлмөгө болгон шилтеменин синтаксисин канаатандырат, бирок ал эстин адресин көрсөтпөй жатат.

123.	Too many symbols – Идентификаторлор өтө эле көп. Программа же программалык модул 64К дан ашык идентификаторлорду баяндап жатат.
124.	Statement part too large – Операторлор бөлөгү өтө эле чоң. Turbo Pascal операторлор бөлөгүнүн өлчөмүн болжолдуу түрдө 24К чоңдугуна чейин чектейт.
126.	Files must be var parameters – Файлдар var параметрлерине ээ болушу керек. Файлдык типтин параметрлери параметр-өзгөрүлмөлөр болушу керек.
127.	Too many conditional symbols – Шарттуу идентификаторлор өтө көп. Шарттуу идентификаторлорду аныктоо үчүн орун жок.
128.	Misplaced conditional directive – Шарттуу директива калып кеткен. Компилятор тиешелүү {$IFDEF}, {$IFNDEF} же {$IFORT} директиваларысыз {$ELSE} же {$ENDIF} директивасы бар экендигин тапты.
129.	ENDIF directive missing – ENDIF директивасы калып кеткен. Баштапкы файлда {$IFxxx} жана {$ENDIF} директивалар барабар санда болушу керек.
130.	Error in initial conditional defines – Баштапкы шарттуу аныктамаларда ката бар.
131.	Header does not match previous definition – Бөрк мурдагы аныктамага тиешелеш келбейт. Процедуранын же функциянын интерфейстик секцияда же forward баяндоосунда көрсөтүлгөн бөркү процедуранын же функциянын өзүнүн бөркүнө тиешелеш келбейт.
132.	Cannot evaluate this expression – Берилген туюнтманы эсептөөгө бобьойт. Туюнтма-константада же оңдоп-түзөө туюнмасында (отладочное выражение) колдонууга мүмкүн болбогон каражаттарды пайдалануга болгон аракет бар.
134.	Expression incorrectly terminated – Туюнтманын корректтүү эмес (тура эмес) аякташы. [Тиркелген оңдоп-түзөгүч (отладчик) үчүн гана]. Turbo Pascal бул жерде туюнтманын бүтүшүн же амалды күтүп жатат, бирок алардын бирөөсүн да таппай жатат.
135.	Invalid format specifier – Форматтын тура эмес спецификатору. [Тиркелген оңдоп-түзөгүч (отладчик) үчүн гана]. Форматтын тура эмес спецификатору пайдаланылган же форматтын спецификаторунун сандык аргументи уруксат берилген чектен чыгып кеткен.
136.	Invalid indirect reference – Уруксат берилбеген кыйыр шилтеме. Оператор уруксат берилбеген кыйыр шилтемени ишке ашырууга аракет кылып жатат.
137.	Stuctured variable are not allowed here – Бул жерде структуралык өзгөрүлмөнү пайдалануга уруксат жок.

138.	Cannot evaluate without System unit – System блогу жок эсептөөгө болбойт. [Тиркелген оңдоп-түзөгүч (отладчик) үчүн гана]. Оңдоп-түзөгүч туюнтманы эсептей алышы үчүн .TPL файлында System модулу кармалып турушу керек.
139.	Cannot access this sumbol – Бул идентификаторго кайрылууга уруксат жок. [Тиркелген оңдоп-түзөгүч (отладчик) үчүн гана]. Программа компиляцияланып бүтөөрү менен андагы бардык идентификаторлордун көптүгүнө кайрылууга мүмкүн болуп калат. Бирок, айрым бир идентификаторлорго программа ишке салынмайынча кайрылууга болбойт.
140.	Invalid floating-point operation – Уруксат берилбеген жылма чектиттүү амал. Эки чыныгы маанилердин үстүнөн болгон амалда ашып кетүү (переполнение) же нөлгө бөлүү болуп калды.
141.	Cannot compile overlay to memory – Оверлейлерди эске компиляциялоону аткарууга болбойт. [Реалдык режим]. Оверлейлерди пайдалануучу программа дискке компиляцияланышы керек.
142.	Procedure or function variable expected – Процедуралык же функционалдык өзгөрүлмө пайдаланылышы керек. Assigned стандарттык процедурасы өзгөрүлмө-көрсөткүч тибиндеги же процедуралык типтеги аргументти талап кылат.
143.	Invalid procedure or function reference – Процедурага же функцияга болгон уруксат берилбеген шилтеме.
144.	Cannot overlay this unit – Бул модул оверлейлик модул катары пайдаланыла албайт. [Реалдык режим]. {$O+} директивасы менен компиляция болбогон модулду оверлейлик модул катары пайдалануга аракет болуп жатат.
145.	Too many nested scopes – Камтылуучулуктун деңгээли өтө чоң. Камтылуучулуктун деңгээлине uses сүйлөмүндөгү ар бир unit, камтылуучулукка ээ болгон ар бир жазуу, with операторлорунун камтылуучулугу таасир этет.
146.	File access denied – Файлга кайрылууга болбойт. Файлды ачууга же түзүүгө болбойт.
147.	Object type expected – Объекттик тип керек. Идентификатор об’екттик типти аныктабай жатат.
148.	Local object ture are not allows – Локалдык об’ екттик типтерди баяндоого уруксат жок.
149.	VIRTUAL expected – VIRTUAL талап кылынат. VIRTUAL кызматчы сөзү жок.
150.	Method identifier expected – Усулдун идентификатору керек. Идентификатор усулдун идентификатору эмес.
151.	Virtual constructor are not allowed – Виртуалдык конструкторго уруксат жок. Конструктордун усулу статикалык болушу керек.

152.	Constructor identifier expected – Конструктордун идентификатору керек. Идентификатор конструктордун идентификатору эмес.
153.	Destructor identifier expected – Деструктордун идентификатору керек. Идентификатор деструктордун идентификатору эмес.
154.	Fail only allowed within constructors – Fail ге конструкторлордун ичинде гана уруксат берилет. File стандарттык процедурасы конструктордун ичинде гана пайдаланыла алат.
155.	Invalid combination of opcode and operands - Амал коду менен операндардын уруксат берилбеген комбинациясы. Ассемблердеги амал коду операндардын мындай айкалышын кабыл албай жатат.
156.	Memory reference expected – Эске болгон шилтеме талап кылынат. Ассемблердин операнды, бул жерде талап кылынгандай, эске болгон шилтеме боло албайт.
157.	Cannot add or sbstract relocatable symbols - Аралашма (перемещаемые) идентификаторлорду кошууга же кемитүүгө болбойт. Ассемблердин операндында аралашма (перемещаемые) идентификаторлордун үстүнөн аткарууга уруксат берилген жалгыз амал - бул константаны кошуу же константаны кемитүү.
158.	Invalid register combination – Регистрлердин уруксат берилбеген айкалышы. Индекстик регистрлердин уруксат берилген айкалышы: [BX], [BP], [SI],[DI],[BX+SI],[BX+DI],[BP+SI] жана [BP+DI]
159.	286/ 287 Instructions not allowed - 286/ 287 процессорлорунун инструкцияларына уруксат жок. Бул көрсөтүлгөн процессорлордун амалдар коддоруна уруксат берүү үчүн компилятордун {$G+} директивасын пайдалануу керек, бирок жыйынтыктоочу код 8086 жана 8088 процессорлуу машиналарда иштей албайт.
160.	Invalid symbol reference – Идентификаторго болгон уруксат берилбеген шилтеме. Берилген идентификаторго ассемблердин операнды үчүн уруксат жок.
161.	Code generation error – Кодду генерациялоо катасы. Оператордун мурда келген бөлүгү керек болгон эн-белгиге (меткага) жете албай жаткан LOOPNE, LOOPE, LOOP же JCXZ инструкцияларын кармап турат.
162.	ASM expected – ASM кызматчы сөзү керек.
163.	Duplicate dynamic method index – Динамикалык усулдун индекси кайталанып жатат. Динамикалык усулдун бул индекси эбак башка усул тарабынан пайдаланылган.
164.	Duplicate resource identifier – Ресурс идентификаторунун кайталанышы. [Windows же корголгон режим үчүн гана]. Берилген файл башка ресурс үчүн эбак пайдаланылган атка же идентификаторго ээ болгон ресурсту кармап турат.

165.	Duplicate or invalid export index – Экспорттун кайталанган же уруксат берилбеген индекси. [Windows же корголгон режим үчүн гана]. Index операторунда берилген иреттик номер 1 .. 32767 диапазонунда жатпайт же ал номер башка экспорттолуучу камтылуучу программа тарабынан эбак пайдаланылган.
166.	Procedure or function indentifier expected – Процедуранын же функциянын идентификатору керек. [Windows же корголгон режим үчүн гана]. Export оператору процедураларды жана функцияларды гана экспорттоого урусат берет.
167.	Cannot export this symbol – Бул идентификаторду экспорттоого болбойт. [Windows же корголгон режим үчүн гана]. Процедура же функция, эгерде ал Export процедурасынын операторунда баяндалбаган болсо, анда экспорттоло албайт.
168.	Duplicate export name – Экспорттолуучу аттын кайталанышы. [Windows же корголгон режим үчүн гана]. Name операторунда берилген ат башка экспорттолуучу камтылуучу программа үчүн эбак пайдаланылган.
169.	Executable file header too large – Аткарылуучу файлдын бөркү өтө эле чоң. [Windows же корголгон режим үчүн гана]. EXE файлдын генерацияланууучу бөркү 64К (компоновщик үчүн жогорку предел) дан ашып кеткен.

Аткаруу убактысынын каталары

Аткаруу убактысынын каталары төмөндөгүдөй форматка ээ.

Runtime error nnn at xxxx:yyyy – xxxx: yyyy адреси боюнча аткаруу убактысынын **nnn** катасы.

DOS тун каталары

1.	Invalid function number – Уруксат берилбеген функциянын номери. DOS тун жок (жашабаган) функциясына кайрылуу.
2.	File not found – Файл табылган жок. Эгер файлдык өзгөрүлмөгө ыйгарылган ат жок (жашабаган) эле файлды көрсөтүп турса, анда Reset, Append, Rename же Erase процедуралары тарабынан ушул ката генерацияланат.
3.	Path not found – Маршрут табылган жок. Эгер файлдык өзгөрүлмөгө ыйгарылган ат жараксыз болсо (недействительный) же жок (жашабаган) камтылуучу каталогду (подкаталогду) көрсөтүп турса, анда Reset, Append, Rwrite же Erase процедуралары тарабынан ушул ката генерацияланат. Ошондой эле маршрут жараксыз болгон (недействительный) же жок (жашабаган) камтылуучу каталогду (подкаталог) көрсөтүп турган учурда ChDir, MkDir же RmDir процедуралары тарабынан ушул ката генерацияланат.

4.	Too many open files – Ачылган файлдардын саны өтө көп.
5.	File access denied – Файлга кайрылууга уруксат жок.
6.	Invalid file handle – Файлды баяндоодо мындай баяндагычка уруксат жок. Бул ката DOS тун системалык чакыруусуна файлдын урукста берилбеген баяндагычы берилгенде генерацияланат.
12.	Invalid file access code – Файлдарга кайрылуу коду жараксыз. Бул ката типтүү же типсиз файлдарды ачууда FileMode маниси жараксыз болгон учурда Reset жана Append процедуралары тарабынан генерацияланат.
15.	Invalid drive number – Диск салгычтын (дисководдун) мындай номерине уруксат жок. Бул ката диск салгычтын номери тура эмес болгон учурда GetDir процедурасы тарабынан генерацияланат.
16.	Cannot remove current directory – Учурдагы каталогду жоготууга (өчүрүүгө) болбойт. Эгерде маршрут учурдагы каталогду көрсөтүп төрган болсо, анда RmDir процедурасы тарабынан ушул ката генерацияланат.
17.	Cannot rename across drives – Кайра атоодо ар түрдүү диск салгычтарды көрсөтүүгө болбойт. Эгерде эки файл тең бир эле дискте турбаса, анда Rename процедурасы тарабынан ушул ката чыгат.
18.	No more files – Файлдар жок. Мындай билдирүү качан FindFirst же FindNest чакыруулары берилген атка же атрибуттардын жыйынына туура келген файлдарды таппай калган учурда Dos модулундагы DosError өзгөрүлмөсү аркылуу чыгарылат.

Кийрүү-чыгаруу каталары

Эгерде {$I+} директивасы менен компиляцияланган операторлордун кайсы биринде кийрүү-чыгаруу катасы болуп жатса, анда ал программанын аткарылышынын токтошуна алып келет. Ал эми эгерде кийрүү-чыгаруу катасы бар оператор {$I-} директивасы кошулган (включенный) абалда компиляцияланса, анда программанын аткарылышы улана берет дагы, катанын номерин IOResult функциясынын жардамында алууга болот.

100.	Disk read error – Дискти окуу катасы.
101.	Disk write error – Дискке жазуу катасы. Эгерде диск толуп калган болсо, анда Close, Write, Writeln, же Page процедуралары тарабынан ушул ката генерацияланат.
102.	File not assigned – Файлга ат берилген эмес.
103.	File not open – Файл ачылган эмес. Эгерде файл ачылбаган болсо, анда Close, Read, Write, Seec, Eof, FilePos, FileSize, Flush, BlockRead же BlockWrite процедуралары тарабынан ушул ката генерацияланат.

104.	File not open for input – Кийрүү үчүн файл ачылган эмес. Эгерде файл кийрүү үчүн ачылбаган болсо, анда Read, Readln, Eof, Eoln, SeecEof же SeecEoln процедуралары тарабынан тексттик файлда ушул ката генерацияланат.
105.	File not open for output – Чыгаруу үчүн файл ачылган эмес. Эгерде файл чыгаруу үчүн ачылбаган болсо, анда Write, Writeln же Page процедуралары тарабынан тексттик файлда ушул ката генерацияланат.
106.	Invalid numeric format – Туура эмес сандык формат. Тексттик файлдан окулган сандык маани туура сандык форматка тиешелеш келбеген учурда Read же Readln процедуралары тарабынан ушул ката генерацияланат.

Критикалык каталар

Критикалык каталар реалдык же корголгон режимде пайда болушу мүмкүн.

150.	Disk is write protected – Диск жазуудан корголгон.
151.	Unknown unit – Белгисиз модуль.
152.	Drive not ready – Диск салгыч «даяр эмес» абалда.
153.	Unknown command – Таанылбаган (неопознанная) команда.
154.	CRC error in data – Берилгендерде ката бар.
155.	Bad drive requiest structure length – Дискке кайрылууда структуранын туура эмес узундугу көрсөтүлгөн.
156.	Disk seek error – Дискте бөркчөлөрдү (головкаларды) коюу (установка) амалы учурундагы ката.
157.	Unknown media type – Ташып жүрүүчүнүн белгисиз тиби.
158.	Sector not found – Сектор табылган жок.
159.	Printer out of paper – Печатточу түзүлүштө кагаз түгөндү.
160.	Device write fault – Берилген түзүлүшкө жазуудагы ката.
161.	Device read fault – Берилген түзүлүштөн окуудагы ката.
162.	Hardware failure – Аппаратуранын таштап жиберүүсү (сбой). Биргелешип кайрылуу бузулган учурда же ар түрдүү тармактык каталар болгондо DOS ушул ката жөнүндөгү билдирүүнү чыгарат.

Фаталдык каталар

Фаталдык каталар программанын ишинин дароо токтошуна алып келет.

200.	Division by zero – Нөлгө бөлүү. Программада /, mod же div амалы учурунда санды нөлгө бөлүү аракети болуп жатат.
201.	Range chek error – Чек араларды текшерүүдөгү ката.
202.	Stac overflow error – Стектин ашып кетиши (переполнение).

203.	Heap overlow error – Эстин динамикалык бөлүштүрүлүүчү аймагынын ашып кетиши (переполнение).
204.	Invalid pointer operation – Жараксыз (недействительная) шилтеме амалы.
205.	Floating point overflow – Жылма чекиттүү (с плавающей точкой) амалды аткарууда ашып кетүү болду. Жылма чекиттүү амал ашып кетүүгө алып келди.
206.	Floating point underflow – Жылма чекиттүү амалды аткарууда иреттин (тартиптин) жоголуп кетиши. Жылма чекиттүү амал иреттин (тартиптин) жоголуп кетишине алып келди. Атайын көрсөтүлбөгөн учурда иреттин (тартиптин) жоголуп кетиши нөлгө барабар болгон жыйынтыкты кайтарып берет.
207.	Invalid floating point operation – Уруксат берилбеген жылма чекиттүү амал.
208.	Overlay manager not installed – Оверлейлерди башкаруучу камтылуучу система (подсистема) орнотулган эмес. [Реалдык режим үчүн гана].
209.	Overlay file read error – Оверлейлелик файлды окуу катасы. [Реалдык режим үчүн гана]. Оверлейлерди башкаруучу камтылуучу система оверлейлелик файлдан оверлейди окууга аракет кылганда окуу катасы пайда болду.
210.	Objec not initialized – Объект инициализацияланган эмес. Диапазонду текшерүү кошулганда объект конструкторду чакыруунун жардамында аныкталганга чейин обекттин виртуалдык усулуна кайрылуу болуп өттү.
211.	Call to abstract method – Абстракттык усулду чакыруу.
212.	Stream registration error – Агымды регистрациялоонун катасы.
213.	Collektion index out of range – Топтомдун индекси диапазондон сырткары. TCollection усулуна бериллүүчү индекс диапазондун чегинен чыгып кеткен.
214.	Colektion overlow error – Топтомдун ашып кеттүү (переполнение) катасы. Мындай ката TCollection тарабынан кеңейтүүгө мүмкүн болбогон топтомго элементти кошууга аракет жасалганда чыгат.
215.	Arithmetic overflow error – Арифметикалык ашып кеттүү (переполнение). Бул ката {$G+} абалында компиляцияланган операторлор тарабынан арифметикалык амал ашып кеттүүгө алып келген учурда чыгарылат.
216.	General Protection fault – Коргоонун жалпы бузулушу. [Корголгон режим үчүн гана].

Корголгон режимдеги (DPMI) DOS интефейсинин каталары

Биз бул жерде программа корголгон режимде аткарылып жаткан учурда алынышы мүмкүн болгон каталар жөнүндөгү билдирүүлөрдү келтиребиз. Мындай каталар 4 категорияга бөлүнөт: орнотуу каталары (ошибки установки), фиктивдик модулдун каталары (фиктивного модуля), аткаруу этабынын администраторунун каталары жана сервердин каталары.

Орнотуу каталары (DPMIINST)

A20 line already enabled, so test is meaningless – A20 тармагы эбак аракетке келтирилген, текшерүү мааниге ээ эмес.

Фиктивдик модул каталары.

Фиктивдик модул – эгерде жок болгон болсо DPMP – сервердик жана аткаруу этабынын администраторунун жүктөлүшбнө жооп берет. Фиктивдик модулдун каталары жөнүндөгү билдирүүлөр төмөндөгүдөй форматка ээ:

Stub error (xxxx): xxxx

мында Stub error ката фиктивдик модул тарабынан генерациялангандыгын көрсөтөт, (xxxx) – ката жөнүндөгү билидирүүнүн номери, ал эми xxxx – билидирүүнүн тексти.

Stub error (0001):	**needs at least 286** – Фиктивдик модулдун катасы: 286 дан кем болбогон процессор керек.
Stub error (2002):	**can`t find rtm.exe -** Фиктивдик модулдун катасы: rtm.exe файлы табылган жок. Аткаруу этабынын администраторунун файлы - rtm.exe маршрут боюнча же учурдагы каталогдо жатышы керек.
Stub error (2003):	**can`t find DPMI16BI.OVL -** Фиктивдик модулдун катасы: **DPMI16BI.OVL** файлы табылган жок. **DPMI16B1.OVL** файлы маршрут боюнча же учурдагы каталогдо жатышы керек.
Stub error (0012):	**file not found -** Фиктивдик модулдун катасы: файл табылган жок. Фиктивдик модул колдонмо программанын файлын таба алган жок.
Stub error (0013):	**path not found -** Фиктивдик модулдун катасы: маршрут табылган жок.
Stub error (0015):	**file access denied -** Фиктивдик модулдун катасы: кайрылууга уруксат берилбеген файл.

Stub error (0018):	**not enough memory to load file -** Фиктивдик модулдун катасы: файлды жүктөө үчүн эсте орун жетишпейт.
Stub error (001A):	**invalid environment -** Фиктивдик модулдун катасы: уруксат берилбеген операциялык чөйрө. DOS системанын чөйрөсүнүн спецификасы бузулган.
Stub error (001B):	**invalid file -** Фиктивдик модулдун катасы: Файлга кайрылууга уруксат жок.
Error:	**no DOS extensions in DPMI server –** Ката: **DPMI** серверде DOS тун кеңейтирилиштери жок. Пайдаланылуучу **DPMI -** сервер тармактык стандарттарды кармабайт.
Error:	**heeds DOS 3.x or higher -** Ката: DOS тун 3.х же андан жогорку версиясы талап кылынат.
Error:	**failed to locate DPMI-server(DPMI16BI.OVL) - DPMI –** серверди табууга мүмкүн болбой жатат. **DPMI16BI.OVL** (корголгон режимдеги колдонмо программа катары) көрсөтүлгөн маршрут боюнча жайгашкандыгына ишенүү керек.

Аткаруу этабы администраторунун каталары

Аткаруу этабы администраторунун каталарынын көпчүлүгү эстин жетишпегендигинен улам пайда болот.

Мындай типтеги каталар төмөндөгүдөй форматка ээ:

Loader error (xxxx): xxxx
(Жүктөөчүнүн катасы (xxxx): xxxx)

мында **Loader error** ката фиктивдик модулдун жүктөөчүсү тарабынан генерациялангандыгын көрсөтөт, **(xxxx)** - катанын номери, ал эми **xxxx –** билдирүүнүн тексти.

Loader error (0001):	**out of memory -** Жүктөөчүнүн катасы: эс жок.
Loader error (0002):	**out of selectors -** Жүктөөчүнүн катасы: селекторлор жок.
Loader error (0003):	**out of internal tables -** Жүктөөчүнүн катасы: ички таблицалар жетишпейт. Жүктөөчү ички таблицалар дын пределдик маанисин жогорулатып жиберди.
Loader error (0020):	**invalid dunamic link-** Жүктөөчүнүн катасы: уруксат берилбеген динамикалык компоновка. DLL ден импорттун уруксат берилбеген шилтемеси.

Loader error (0022):	**could't open file-** Жүктөөчүнүн катасы: файлды ачууга мүмкүн эмес. Файл же файл тарабынан импорттолуучу DLL табылган жок же ачууга болбой жатат.
Loader error (0023):	**invalid exe format -** Жүктөөчүнүн катасы: .exe – файлдын уруксат берилбеген форматы. Файл же файл тарабынан импорттолуучу DLL табылган жок же уруксат берилбеген форматка ээ.
Loader error (0024):	**wrong version -** Жүктөөчүнүн катасы: туура эмес версия. **DPMI16.OVL** файлы корректтүү маршрут боюнча жайгашкан жана жүктөөгө уруксат берилген биринчи файл болушу керек.
Loader error (0025):	**cannot initialize -** Жүктөөчүнүн катасы: инициализа циялоого мүмкүн эмес.
Loader error (0026):	**DLL initialization error -** Жүктөөчүнүн катасы: DLL ди инициализациялоо катасы. Инициализациялоо камтылуучу программаларынын бирөөсү тура эмес аяктады (башкача айтканда катанын нөл эмес кодун кайтарып берди).
Error:	**error the environment string –** Ката: операциялык чөйрөнүн жолчосундагы ката. "RTM" операциялык чөйрөсүнүн жолчосундагы корректтүү эмес параметрлер.
Runtime error:	**invalid entry point called –** Аткаруу этабынын катасы: кирүүнүн уруксат берилбеген чекитин чакыруу. Колдонмо программа жок эле атка же модулдардын бирөөсүнүн иреттик номерине шилтеме жасап жатат.
Application errors:	**Application load & execute error 0001; Application load & execute error FFF0 -** Колдонмо программанын жүктөлүш жана аткарылыш катасы. Корголгон режимдеги колдонмо программаны жүктөө үчүн эс жетишпейт.

DPMI - сервердин каталары

DPMI-сервердин каталары жөнүндөгү билдирүүлөр Borland DPMI-сервер тарабынан гана генерацияланып экранда төмөндөгүдөй формата пайда болот:

DPMI error (xxxx): xxxx

мында **DPMI error -** ката DPMI – сервер тарабынан генерацияланган-дыгын көрсөтөт, **(xxxx)** - катанын номери ал эми **xxxx –** билдирүүнүн тексти.

DPMI error (4001):	**insufficient memory for initialization –** DPMI нин катасы: инициализациялоо үчүн эс же тишпейт. Серверди ишке салуу (запуск) үчүн эс жетишпейт.
DPMI error (4002):	**memory manager does not support DPMI or VCPI–** DPMI нин катасы: эстин администратору DPMI ни же VCPI ни кармабай жатат (не поддерживает). DPMI же VCPI табылган жок.
DPMI error (4004):	**unrecognized hardware, run DPMINST -** DPMI нин катасы: тааныш эмес аппаратура, **DPMIINST** ти ишке салгыла. DPMI-сервер сиз пайдаланган аппаратураны тааныбай жатат.
DPMI error (4005):	**unrecognized environment parameters -** DPMI нин катасы: операциялык чөйрөнүн таанылбаган параметрлери. DPMIMEM операциялык чөйрөсүнүн өзгөрүлмөсүнүн параметрлери корректтүү эмес (тура эмес).
DPMI error (4007):	**bad A20 off parameter -** DPMI нин катасы: туура эмес A20 пармаетри. DPMI-сервер A20 параметри менен корректтүү (туура) иштей албайт. HIMEM.SYS эс администратору туура (корректно) орнотулган болушу керек.
DPMI error (4008):	**bad A20 on parameter -** DPMI нин катасы: туура эмес A20 пармаетри.
DPMI error (4009):	**bad switch parameter -** DPMI нин катасы: туура эмес пармаетр-кайра туташтыргыч.XMMDOS, XIMEM.SYS же QEMM драйверинин тура эмес иштеши.
DPMI error (4009):	**insufficient extended memory -** DPMI нин катасы: кеңейтирилген эс жетишпейт. Сервердин иши үчүн кеңейтирилген эс жетишпейт.
DPMI error (400C):	**cannot create linear address space -** DPMI нин катасы: сызыктуу адрестик мейкиндикти түзүү мүмкүн эмес.
DPMI error (400D):	**cannot create system address space -** DPMI нин катасы: системалык сызыктуу мейкиндикти түзүү мүмкүн эмес.
DPMI error (400E):	**cannot copy kernel to high memory -** DPMI нин катасы: ядрону эстин чоң адрестерине көчүрүү мүмкүн эмес.

DPMI error (400F):	**undefined error -** DPMI нин катасы: аныкталбаган ката.
DPMI error (4010):	**unable to copy DOSX –** DPMI нин катасы:**DOSX** ти көчүрүп алуу мүмкүн эмес.
DPMI error (4011):	**unable to copy IDT -** DPMI нин катасы: **IDT** тини көчүрүп алуу мүмкүн эмес.
DPMI error (4012):	**unable to create int chain table -** DPMI нин катасы: үзгүлтүктөр чынжырчасынын таблицасын түзүү мүмкүн эмес
DPMI error (4013):	**unable to create PM stack -** DPMI нин катасы: корголгон режим үчүн стекти түзүү мүмкүн эмес
DPMI error:	**Bad environment params –** Операциялык чөйрөнүн тура эмес параметрлери. Корректтүү синтаксисти пайдаланып DPMIMEM операциялык чөйрөсүнүн згөрүлмөсүнүн маанисин кайрадан бергиле.
DPMI error:	**Machine not in database (run DPMIINST) –** Машина берилгендер базасында жок болуп жатат, DPMIINST ти аткаргыла. DPMI-сервер ядронун берилгендер базасында издөөнү аткарып жатат, сиздин машинаңыз жөнүндө информацияны таппай жатат.
DPMI error :	**Not enough memory for PM init –** Корголгон режимди инициализациялоо үчүн эс жетишпейт. DPMI-сервер корголгон режимди инициализация-лашы үчүн эс жетишпей жатат.
DPMI error:	**V86 task without vcpi – VCPI** жок **V86** маселеси. DPMI - сервердин корголгон режимге өтүшүнө мүмкүнчүлүк бербей жаткан башка маселе аткарылып жатат.

2 – ТИРКЕМЕ. КОМПИЛЯТОРДУН ДИРЕКТИВАЛАРЫ

Turbo Pascal тилинин чөйрөсүндө программалоодо компилятордун директиваларын сабаттуу пайдалана билүү башкы мааниге ээ болот. Тилдин башка элементтери менен компилятордун директиваларынын ортосунда принципиалдуу айырмачылыктар бар. Директивалар компилятордун же компоновщиктин (микропроцессордун эмес) иштөөсүн башкарат. Алар компиляторго, мисалы, берилгендерди түздөө (выравнивание), пайдаланылуучу процедуралар менен функцияларды чакыруу тибин ж.б. берет. Мындай директивалардын бир топторун биз карап өткөн мисалдарда пайдаландык. Төмөндө Turbo Pascal тилинин компиляторунун директиваларынын тизмегин алардын арналыштары менен кошо келтиребиз.

{$A+} же {$A-} директивасы

Атайын көрсөтүлбөгөн учурда {$A+} директивасы аракет этет.

Бул директива өзгөрүлмөлөр менен типтештирилген константаларды сөздүн чегине түздөөдөн байттын чегине түздөөгө жана тескерисинче өтүүгө мүмкүнчүлүк берет.

{$A+} абалында өлчөмү боюнча бир байттан ашып кеткен өзгөрүлмөлөр менен типтештирилген константалар машиналык сөздүн (жуп манилүү адрестер) чегине түздөлөт.

{$A-} абалында түздөөгө байланышкан эч кандай амал каралбайт.

{$B+} же {$B-} директивасы

Атайын көрсөтүлбөгөн учурда {$B+} директивасы аракет этет.

Бул директивалар бульдук **and** жана **or** амалдары үчүн кодду генерациялоонун эки ар түрдүү түрүн ишке ашырат.

{$B+} абалында компилятор бульдук туюнтманы толук эсептөө үчүн кодду генерациялайт.

{$B-} абалында компилятор бульдук туюнтманы кыска схема боюнча эсептөө үчүн кодду генерациялайт.

{$C атрибут атрибут…}

Көрсөтүлбөгөн учурда {$C MOVEABLE DEMANDLOAD DISCARDABLE}

Бул директива код сегментинин атрибуттарын (Windows жана корголгон режим үчүн гана)башкаруу үчүн пайдаланылат.

Колдонмо программадагы же библиотекадагы ар бир код сегменти аны эске жүктөгөн учурда ал сегмент өзүн кандай алып жүрөөрүн аныктай турган атрибуттардын жыйындысына ээ болот.

$C директивасы өзү жайгашкан программалык модулдун (модулдун, программанын же библиотеканын) ичиндеги код сегментине гана таасир эте алат.

{$D+} же {$D-} директивасы

Атайын көрсөтүлбөгөн учурда {$D+} директивасы аракет этет.

Бул директива оңдоп-түзөө (отладка) үчүн информацияны генерациялоону берет же алаып салат.

Эгерде программа же модул **{$D+}** абалында компиляцияланса, анда Turbo Pascal дын тиркелген оңдоп-түзөгүчү бизге бул модулду кадамдап аткарууга мүмкүнчүлүк берет жана ал модулда аяктоо чекитин коңт.

{$DEFINE **ат**} директивасы

$DEFINE директивасы берилген атка ээ болгон шарттуу идентификаторду аныктайт. Идентификатор баштапкы компиляциялануучу коддун калган бөлүгү үчүн же ал {$UNDEFINE **ат** } директивасында пайда болгонго чейин аныкъалган болот. Эгерде «ат» эбак аныкталган болсо, анда {$DEFINE **ат**} директивасы башка эч кандай аракеттерди пайда кылбайт.

{$D **текст**} директивасы

$D директивасы пайдалануучу тарабынан берилген текстти .EXE же .DLL файлынын бөркүндөгү файлдын баяндалыш жазуусуна сыйлыгыштырат.

{$ELSE} директивасы

Атайын көрсөтүлбөгөн учурда {$C+} **директивасы аракет этет.**

ELSE директивасы акыркы {$IFxxx}менен чектелген жана {$ENDIF}тен кийинки баштапкы текстти компиляциялоого же четтетүүгө алып келет.

{$E+} же {$E-} директивасы

Атайын көрсөтүлбөгөн учурда {$E-} **директивасы аракет этет.**

Эмуляция 8087 процессорлошу (соопроцессору) жок болгон учурда анын ишин эмуляциялоочу аткаруучу системанын библиотекасы менен компоновкага уруксат берет же тыйуу салат.

{$N+ E-}абалында Turbo Pascal тили 8087 процессорлошу болгон учурда гана пайдаланылышы мүмкүн болгон программаны түзө алат.

{**$**ENDIF} **директивасы**

ENDIF директивасы акыркы **{$IFxxx}** директивасы мнен башталган шарттуу компиляцияны аяктайт.

{$F+} **же** {$F-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$F+} **директивасы аракет этет.**

Бул директива удаалаш компиляцияланууучу процедуруларды жана функцияларды чакыруу тибин башкарат. {$F+} абалында компиляцияланган процедуралар жана функциялар дайыма чакыруунун алыскы тибин пайдаланышат. {$F-} директивасын көрсөткөн учурда Turbo Pascal тили автоматтык түрдө кайрылуунун тиешелүү тибин тандайт: эгерде процедура же функция программалык бирдиктин интерфейс бөлүгүндө баяндалган болсо анда алыскы (far) тибин, андай болбогон учурда жакынкы (near) тибин тандап алат.

{$G+} **же** {$G-} **директивасы**

Атайын көрсөтүлбөгөн учурда:
(реалдык режим) {$G-}
(корголгон режим жана windows) {G+}
директивасы аракет этет

$G директивасы 80286 процессору үчүн кодду генерациялоого уруксат берет же тыйуу салат. **{$G-}** абалында 8086 процессорунун инструкцияларя гана генерацияланат, жана ошондой эле ушул абалда генерацияланган программалар 80x86 тобундагы каалаган башка процессордо аткарыла алат.

{ $G **модулдун аты, модулдун аты**…} **директивасы**

$G директивасы бизге компоновщик бир сегменттин ичинде жайгаштырышы керек болгон модулдардын тайпаларын берүүгө мүмкүнчүлүк берет. (Windows жана корголгон режим үчүн гана).

Ар бир G директивасы модулдардын тайпасын берет. $G директивасына программанын же библиотеканын uses операторунда гана уруксат берилет.

Код сегментинин атрибуттарын $C директивасынын жардамында башкарууга болот. Сегменттин уруксат берилген өлчөмү $S директивасы менен коюлат.

{ $IFDEF **индентификатор** } **директивасы**

IFDEF директивасы эгерде “идентификатор” аты аныкталган болсо, анда кийин келүүчү баштапкы текстти компиляциялайт.

{ $IFNDEF индефикатор } **директивасы**

IFNDEF директивасы эгерде “идентификатор” аты аныкталбаган болсо, анда кийин келүүчү баштапкы текстти компиляциялайт.

{$IFORT **кайра_туташтыргыч} директивасы**

Эгерде берилген маалда “кайра_туташтыргыч” көрсөтүлгөн абалда турса, анда IFORT директивасы кийин келүүчү баштапкы текстти компиляциялайт. Кайра_туташтыргыч (директива) өзүнөн кийин плюс (+) же минус (-) белгиси келүүчү директива - кайра_туташтыргычтын атынан турат.

{$I **файлдын_аты**} **директивасы**

Бул директива компиляторго компиляциялоого аталган файлды кошуу зарылдыгын билдирет. Иш жүзүндө файл компиляцияланган текстке түздөн түз {$I **файлдын_аты**} директивасынан кийин сыйлыгыштырылып коюлат.

{$I+} же {$I-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$i+} директивасы аракет этет.

Бул директива кийрүү-чыгаруу процедурасына кайрылуунун жыйынтыгын текшерүү кодун генерациялоону берет же алып салат.

{$K+} же {$K-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$K+} директивасы аракет этет

$K директивасы тиркеме тарабынан экспорттолуучу процедураларды жана функцияларды эффективдүү чакыруу генерациясын башкарат (Windows үчүн гана). Эгерде колдонмо (прикладдык) программа {$K-} абалында генерацияланса, анда

чакырылуучу API Windows камтылуучу программаларын түзүүдө ал MakeProcInstance жана FreeProcInstance камтылуучу программаларын пайдаланышы керек. Атайын көрсөтүлбөгөн учурдагы {$K+} абалында колдонмо (прикладдык) программа өзү экспорттолгон кийрүү чекиттерин чакыра алат жана MakeProcInstance жана FreeProcInstance камтылуучу программаларын пайдалануу зарылчылыгы жок болот.

{$L **файлдын_аты** } **директивасы**

Бул директива компиляторго көрсөтүлгөн файлды компиляцияланууучу программа же программалык модул менен компоновкалоого көрсөтмө берет. $L директивасы сырткы (**external**) камтылуучу программа катары баяндалган камтылуучу программалар үчүн ассемблер тилинде жазылган кодду компоновкалоо үчүн пайдаланылат.

{$L+} же {$L-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$L+} директивасы аракет этет.

Бул директива локалдык идентификаторлор жөнүндөгү информацияны генерациялоону аракетке келтирет же алып салат.

(реалдык режим)
{$M стек_өлчөмү, динам_обл_мин_өлчөмү, динам_обл_макс_өлчөмү}
(корголгон режим)
{$M стек_өлчөмү}
(Windows)
{$M стек_өлчөмү, динам_обл_өлчөмү } директивалары

Атайын көрсөтүлбөгөн учурда
(реалдык режим)
{$M 16384,0, 655360 }
(корголгон режим)
{$M 16384}
(Windows)
{$M 8192,8192}

Бул директива программанын эсин бөлүштүрүү параметрлерин көрсөтөт.

$M директивасы модулдарда пайдаланылса анда ал четтетилет.

{$N+} же {$N-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$N-} директивасы аракет этет.

Бул директива Turbo Pascal да бар болгон жылма чекиттүү эсептөөлөрдүн кодун генерациялоодогу эки моделдин бирөөсүн тандоону ишке ашырат.

{$N-} директивасын көрсөткөн учурда Turbo Pascal аткаруучу системасынын библиотекасынын программаларын чакыруу жардамында программалык камсыздандыруудагы бардык real тибиндеги эсептөөлөрдү аткаруу үчүн кодду генерациялайт.

{$N+} директивасын көрсөткөн учурда бардык real тибиндеги эсептөөлөрдү 8087 математикалык процессорлоштун (соопроцессордун) жардамында аткаруу үчүн код генерацияланат.

{$O+} же {$O-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$O-} директивасы аракет этет.

$O директивасы оверлейлик кодду генерациялоого уруксат берет же тыюу салат (реалдык режим үчүн гана). Turbo Pascal модул {$O+} директивасы менен компиляцияланган болгондо гана аны оверлейлик модул катары пайдаланууга уруксат берет.

Компилятордун {$O+} директивасы дээрлик дайыма {$F+} директивасы менен бирге көрсөтүлүп ал чакыруулардын алыскы (far) тибин пайдаланууга карата оверлейлер администраторунун талаптарын аткарууга мүмкүнчүлүк берет.

{$O модулдун_аты} **директивасы**

{$O модулдун_аты} директивасы анны модулда пайдаланган кезде ал аракет этпейт, ал эми программаны компиляциялаган учурда .EXE файлдын ордуна программадагы пайдаланылуучу кайсы модулдарды .OVR файлга жайгаштыруу керек экендигин берет (реалдык режим үчүн гана).

{$O модулдун_аты} директивалары программанын **uses** опрераторунан кийин көрсөтүлүшү керек.

{$P+} **же** {$P-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$P-} директивасы аракет этет.

$P директивасы **string** кызматчы сөзүнүн жардамында баяндалган параметр-өзгөрүлмөлөрдүн маанисин башкарат. {$P-} абалында **string** кызматчы сөзү менен баяндалган параметр-өзгөрүлмөлөр нормалдуу параметрлер болушат, ал эми {$P+} абалында алар ачык жолчолук параметрлер болушат.

{$Q+} **же** {$Q-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$Q-} директивасы аракет этет.

$Q директивасы ашып кетүүнү (переполнение) текшерүү кодун генерациялоону башкарат. {$Q+} абалында айрым бир арифметикалык амалдар (+, -,*, Abs, Sqr, Succ, Жана Pred) ашып кетүүгө текшерилет.

{$Q+} директивасы стандарттык Inc жана Des директиваларына таасир этпейт. Бул процедуралар эч качан ашып кетүүгө текшерилбейт.

Адата $Q кайра туташтыргычы диапазонду текшерүү кодун генерациялоого уруксат берүүчү же тыйуу салуучу $R кайра туташтыргычы менен айкалышта пайдаланылат.

{$R+} же {$R-} директивасы

Атайын көрсөтүлбөгөн учурда {$R-} директивасы аракет этет.

Бул кайра туташтыргыч чек араны текшерүү кодун генерацялоону аракетке келтирет же алып салат. {$R+} директивасы көрсөтүлгөн учурда индекстелген жолчолуу бардык туюнтмалардын жана массивдердин көрсөтүлгөн чек аранын ичинде жайгашуусу текшерилет, ал эми камтылуучу диапазондордун скалярдык чоңдуктарына жана өзгөрүлмөлөрүнө болгон бардык ыйгаруу операторлорунун берилген чек арада жатышы текшерилет.

Эгерде $R директивасы кошулган болсо (включена), анда виртуалдык усулдарга болгон бардык кайрылуулар чакырууну аткаруучу об’ектин экземпляры үчүн инициализациялоо абалына текшерилет.

Мындай чек араны текшерүүгө жана виртуалдык усулдарга уруксат берүү программалардын аткарылышын секиндетет жана алардын өлчөмдөрүн бир топ чоңойтуп жиберет ошондуктан {$R+} директивасын программаны оңдоп-түзөө (отладка) үчүн гана пайдалануу керек.

{ $R **файлдын_аты** } директивасы

$R директивасы колдонмо программага же библиотекага кошулушу керек болгон ресурстар файлынын атын берет (Windows жана корголгон режим үчүн гана). Көрсөтүлгөн файл Windows дун ресурстар файлы болушу керек. Көрсөтүлбөгөн учурда ал .RES кеңейтирилишине ээ болот.

{$S **сегменттин_өлчөмү**} директивасы

Атайын көрсөтүлбөгөн учурда {$S 16384} директивасы аракет этет.

$S директива-параметрине негизги программада же библиотекада гана уруксат берилет (Windows жана корголгон режим

үчүн гана). Бул директива модулдарды тайпалоо үчүн код сегментинин ылайыктуу деп эсептелген өлчөмүн берет. Көрсөтүлгөн өлчөм 0..65535 диапазонунда жатышы керек. Мындай өлчөмдөн ашып кеткен модулдар өздөрүнүн менчик код сегменттеринде жайгашышат.

$S директивасы эч качан каталар жөнүндөгү эскертүүлөрдү же билдирүүлөрдү бербейт. Эгерде модулдарды башка модулдар менен бирге код сегментинде жайгаштырууга болбосо, анда ал автоматтык түрдө өзүнчө сегментке жайгашат.

{$S+} **же** {$S-} **директивасы**

Атайын көрсөтүлбөгөн учурда {$S+} директивасы аракет этет.

Бул директива стектин ашып кетишин текшерүү менен кодду генерациялоону аракетке келтирет же алып салат.

{$S+} директивасы көрсөтүлгөн учурда компилятор ар бир процедуранын же функциянын башталышында стекте локалдык өзгүрүлмөлөр үчүн жетишерлик орун бөлүнгөнбү же жопу ошону текшерүүчү кодду генерациялайт.

{$T+} **же** {$T-} **директивасы**

Атайын көрсөтүлбөгөн учурда { $T- } директивасы аракет этет.

$T директивасы @ амалы тарабынан генерацияланууучу көрсөткүчтөрдүн маанилеринин типтерин башкарат.

{$T-} абалында @ амалынын жыйынтыгынын тиби ар дайым башка бардык көрсөткүчтөрдүн типтери менен биргелешүүчү типтештирилбеген көрсөткүч болот. Качан {$T+} абалында өзгөрүлмөгө болгон шилтемеге @ амалы колдонулганда жыйынтыктын тиби ^T болот, мында T өзгөрүлмөнүн тибине болгон көрсөткүчтөр менен гана биргелешүүчү.

{ $UNDEF **ат** } **директивасы**

$UNDEF директивасы мурда аныкталган шарттуу идентификаторду алып салат.

{$V+} **же** {$V-} **директивасы**

Атайын көрсөтүлбөгөн учурда { $V+ } директивасы аракет этет.

$V директивасы жолчолорду параметр – өзгөрүлмөлөр катары берүүдө типти текшерүүнү башкарат.

{$V+} абалында формалдык жана иш жүзүндөгү параметрлер окшош (идентичные) жолчолук типтерге (***string***) ээ болушу талап кылынган типти катуу текшерүү аткарылат.

{SV-} абалында иш жүзүндөгү параметр катары жолчолук типтеги каалагандай өзгөрүлмөнү пайдаланууга уруксат берилет, бул

ал өзгөрүлмөнүнүн баяндалган узундугу тиешелүү формалдык параметрдин узундугу менен дал келбеген учурда да тура болот.

{$W+} же {$W-} директивасы

Атайын көсөтүлбөгөн учурда {SW+} директивасы аракет этет.

SW директивасы Windows үчүн спецификалык болуп эсептелген чакыруунун алыскы тибине (far) ээ болгон процедуралар жана функциялар үчүн кирүү жана чыгуу кодун генерациялайт (Windows үчүн гана). {SW+} абалында **far** процедуралар жана функциялар үчүн коддун жана чыгуунун атайын жазуусу генерецияланат. Анын натыйжасында Windows дун реалдык режиминин эсин башкаруучу камтылуучу система (подсистема) код сегменти же берилгендер сегменти жылышкандан кийин чакыруулардын чынжырчасына карата тууралоодо (настройка) стектин алыскы кадрларын корректтүү (туура) идентификацялай алат.

{SW-} абалында кирүүнүн же чыгуунун кошумча жазуусу генерацияланат.

{$X+} же {$X-} директивасы

Атайын көрсөтүлбөгөн учурда {SX+} директивасы аракет этет.

Компилятордун $X директивасы Turbo Pascal дын кеңейтирилген синтаксисине уруксат берет же тыюу салат.

{$X+} режимиминде функцияларды чакыруларды операторлор катары пайдалануга болот, башкача айтканда функциянын жыйынтыгы таштalып жиберилиши мүмкүн.

{$X+} директивасы тиркелген функцияларга колдонулбайт (б.а. System модулунда аныкталган функцияларга).

{$Y+} же {$Y-} директивасы

Атайын көрсөтүлбөгөн учурда {$Y+} директивасы аракет этет.

$Y директивасы шилтемелик информациялардын идентификаторлору үчүн генерацияга уруксат берет же тыюу салат. Программа же модул {$Y+} абалында компиляцияланган учурда Turbo Pascal дын тиркелген карап чыгуу (просмотр) каражаты ушул модул үчүн идентификатордун аныктамаларын жана шилтемелик информацияны чыгара алат.

Адатта $Y директивасы оңдоп-түзөө информацияларын жана локалдык өзгөрүлмөлөр жөнүндөгу информацияларды генерациялоону башкаруучу **$D** жана **$L** кайра туташтыргычтары менен айкалышта пайдаланылат. Эгерде **$D** жана **$L** директиваларына уруксат берилбесе, анда $Y директивасы таасир этпейт.

3 – ТИРКЕМЕ. ASCII КОДУНУН СИМВОЛДОР ТАБЛИЦАСЫ

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	0		32	20		64	40	@	96	60	`
1	1		33	21	!	65	41	A	97	61	a
2	2		34	22	“	66	42	B	98	62	b
3	3		35	23	#	67	43	C	99	63	c
4	4		36	24	$	68	44	D	100	64	d
5	5		37	25	%	69	45	E	101	65	e
6	6		38	26	&	70	46	F	102	66	f
7	7		39	27	‘	71	47	G	103	67	g
8	8		40	28	(	72	48	H	104	68	h
9	9		41	29	)	73	49	I	105	69	i
10	A		42	2A	*	74	4A	J	106	6A	j
11	B		43	2B	+	75	4B	K	107	6B	k
12	C		44	2C	,	76	4C	L	108	6C	l
13	D		45	2D	D	77	4D	M	109	6D	m
14	E		46	2E	.	78	4E	N	110	6E	n
15	F		47	2F	/	79	4F	O	111	6F	o
16	10		48	30	0	80	50	P	112	70	p
17	11		49	31	1	81	51	Q	113	71	q
18	12		50	32	2	82	52	R	114	72	r
19	13		51	33	3	83	53	S	115	73	s
20	14		52	34	4	84	54	T	116	74	t
21	15		53	35	5	85	55	U	117	75	u
22	16		54	36	6	86	56	V	118	76	v
23	17		55	37	7	87	57	W	119	77	w
24	18		56	38	8	88	58	X	120	78	x
25	19		57	39	9	89	59	Y	121	79	y
26	1A		58	3A	:	90	5A	Z	122	7A	z
27	1B		59	3B	;	91	5B	[	123	7B	{
28	1C		60	3C	<	92	5C	\	124	7C	|
29	1D		61	3D	=	93	5D	]	125	7D	}
30	1E		62	3E	>	94	5E	^	126	7E	~
31	1F		63	3F	?	95	5F		127	7F	
128	80	А	160	A0	а	192	C0		224	E0	р
129	81	Б	161	A1	б	193	C1	⊥	225	E1	с

(уландысы)

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
130	82	В	162	A2	в	194	C2	┬	226	E2	т
131	83	Г	163	A3	г	195	C3	├	227	E3	у
132	84	Д	164	A4	д	196	C4	─	228	E4	ф
133	85	Е	165	A5	е	197	C5	┼	229	E5	х
134	86	Ж	166	A6	ж	198	C6	╞	230	E6	ц
135	87	З	167	A7	з	199	C7	╟	231	E7	ч
136	88	И	168	A8	и	200	C8	╚	232	E8	ш
137	89	Й	169	A9	й	201	C9	╔	233	E9	щ
138	8A	К	170	AA	к	202	CA	╩	234	EA	ъ
139	8B	Л	171	AB	л	203	CB	╦	235	EB	ы
140	8C	М	172	AC	м	204	CC	╠	236	EC	ь
141	8D	Н	173	AD	н	205	CD	═	237	ED	э
142	8E	О	174	AE	о	206	CE	╬	238	EE	ю
143	8F	П	175	AF	п	207	CF	╧	239	EF	я
144	90	Р	176	B0	░	208	D0	╨	240	F0	Ё
145	91	С	177	B1	▒	209	D1	╤	241	F1	ё
146	92	Т	178	B2	▓	210	D2	╥	242	F2	Ґ
147	93	У	179	B3	│	211	D3	╙	243	F3	ґ
148	94	Ф	180	B4	┤	212	D4	╘	244	F4	Є
149	95	Х	181	B5	╡	213	D5	╒	245	F5	є
150	96	Ц	182	B6	╢	214	D6	╓	246	F6	І
151	97	Ч	183	B7	╖	215	D7	╫	247	F7	і
152	98	Ш	184	B8	╕	216	D8	╪	248	F8	Ї
153	99	Щ	185	B9	╣	217	D9	┘	249	F9	ї
154	9A	Ъ	186	BA	║	218	DA	┌	250	FA	Ў
155	9B	Ы	187	BB	╗	219	DB	█	251	FB	ў
156	9C	Ь	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	Э	189	BD	╜	221	DD	▌	253	FD	²
158	9E	Ю	190	BE	╛	222	DE	▐	254	FE	■
159	9F	Я	191	BF	┐	223	DF	▀	255	FF	

4 – ТИРКЕМЕ. КЛАВИАТУРАНЫН КОДДОР ТАБЛИЦАСЫ

Клавиатуранын кеңейтирилген коддору

Код	Мааниси	Код	Мааниси
16	Alt+Q	90	Shift+F7
17	Alt+W	91	Shift+F8
18	Alt+E	92	Shift+F9
19	Alt+R	93	Shift+F10
20	Alt+T	94	Ctrl+F1
21	Alt+Y	95	Ctrl+F2
22	Alt+U	96	Ctrl+F3
23	Alt+I	97	Ctrl+F4
24	Alt+O	98	Ctrl+F5
25	Alt+P	99	Ctrl+F6
30	Alt+A	100	Ctrl+F7
31	Alt+S	101	Ctrl+F8
32	Alt+D	102	Ctrl+F9
33	Alt+F	103	Ctrl+F10
34	Alt+G	104	Alt+F1
35	Alt+H	105	Alt+F2
36	Alt+J	106	Alt+F3
37	Alt+K	107	Alt+F4
38	Alt+L	108	Alt+F5
44	Alt+Z	109	Alt+F6
45	Alt+X	110	Alt+F7
46	Alt+C	111	Alt+F8
47	Alt+V	112	Alt+F9
48	Alt+B	113	Alt+F10
49	Alt+N	114	Ctrl+PrtScr
50	Alt+M	115	Ctrl+ ¬
59	F1	116	Ctrl+ ®
60	F2	117	Ctrl+End
61	F3	118	Ctrl+PgUp
62	F4	119	Ctrl+Home
63	F5	120	Alt+1
64	F6	121	Alt+2
65	F7	122	Alt+3
66	F8	123	Alt+4
67	F9	124	Alt+5
68	F10	125	Alt+6
71	Home	126	Alt+7

(уландысы)

Код	Мааниси	Код	Мааниси
72		127	Alt+8
73	PgDn	128	Alt+9
75	¬	129	Alt+0
77	®	130	Alt+ -
79	End	131	Alt+ =
80	¯	132	Ctrl+PgDn
81	PgUp	133	F11
82	Ins	134	F12
83	Del	135	Shift+F11
84	Shift+F1	136	Shift+F12
85	Shift+F2	137	Ctrl+F11
86	Shift+F3	138	Ctrl+F12
87	Shift+F4	139	Alt+F11
88	Shift+F5	140	Alt+F12
89	Shift+F6		

Клавиатуранын сурамжылоо (опрос) коддору

Клавиша	Сурамжылоо коду	Клавиша	Сурамжылоо коду	Клавиша	Сурамжылоо коду
Esc	01	CTR	1D	Пробел	39
! 1	02	A	1E	Caps Lock	3A
@ 2	03	S	1F	F1	3B
# 3	04	D	20	F2	3C
$ 4	05	F	21	F3	3D
% 5	06	G	22	F4	3E
^ 6	07	H	23	F5	3F
&7	08	J	24	F6	40
* 8	09	K	25	F7	41
(9	OA	L	26	F8	42
)0	OB	;	27	F9	43
_ -	OC	”	28	F10	44
+=	OD	~ `	29	F11	D9
BS	OE	Сол Shift	2A	F12	DA
Tab	OF	/	2B	Home	47
Q	10	Z	2C	↑	48

(уландысы)

Клавиша	Сурамжы-лоо коду	Клавиша	Сурамжы-лоо коду	Клавиша	Сурамжы-лоо коду
W	11	X	2D	Pg Up	49
E	12	C	2E	Серый-	4A
R	13	V	2F	←	4B
T	14	B	30	5	4C
Y	15	N	31	→	4D
U	16	M	32	Серый+	4E
I	17	< ,	33	End	4F
O	18	> .	34	↓	50
P	19	? /	35	Pg Dn	51
{[	1A	Оң Shift	36	Ins	52
]}	1B	Prtscr	37	Del	53
Enter	1C	Alt	38	Num Lock	45

АДАБИЯТТАР

1. Абрамов В.Г., Трифонов Н.П. Трифонова Г.Н. Введение в язык Паскаль. – М.: Наука Гл. ред. физ.- мат. лит., 1988. – 320 с.
2. Абрамов С.А., Г.Г. Гнездилова, Е.Н.Капустина, М. И.Селюн. Задачи по программированию. – М.: Наука. гл. ред. физ-мат. лит., 1988. – 224 с.
3. Бердиев А., А.Исраилов, Р.Табышев, Э.Өсөрөв. Компьютер: колдонуучу, программалоо (DOS, BASIC, Turbo Pascal) Бишкек, 2000. – 316 б.
4. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. – М.:МИР, 1998. – 360 с.
5. Гордеев А.В., А.Ю.Молчанов. Системное программное обеспечение. Учебник, Санкт-Петербург-Москва-Харьков-Минск, 2002. – 700 с.
6. Епанешников А., Епанешников М. Программирование в среде Turbo Pascal 7.0 – 4-е изд., испр. и дополн. – М.: Диалог-МИФИ, 2000. – 367 с.
7. Климова Л.М., Практическое программирование. Решение типовых задач. Pascal 7.0, «Кудиц-Образ» Москва, 2000. – 515 с.
8. Марченко А.И., Марченко Л.А. Программирование в среде Turbo Pascal 7.0, Под. ред. Тарасенко В.П. – К.: ВЕК+, М.: Бином Универсал, 1998. – 496 с.
9. Немнюгин С.А., Turbo Pascal. Учебник, Санкт-Петербург-Москва-Харьков-Минск, 2000. – 491 с.
10. Өмүралиев А., Маалыматтар технологиясы, Бишкек, КТМУ, 2002. - 293 б.

Осмоналиев А.Б.
Аркабаев Н.К.

Borland Pascal 7.0

Программалоонун негиздери

1-БӨЛҮК

Терүүгө 09.11.2008-ж. кол коюлду.
Көлөмү 16 басма табк. Нускасы 500. Буюртма №1581.
Ош ш. Курманжан Датка – 209. «Ошоблбасмакана» АК.