Садржај

Методички приручник за наставнике

наставни предмети:

- Рачунарство и информатика
- Увод у програмирање и
- Програмски језици

Овај приручник представља покушај приближавања наставе програмирања ученицима средње школе. Настао као резултат дугогодишњег рада два професора информатике, приручник је током година претрпео велики број измена. Верзија која је сада корисницима на располагању, састоји се од уводног дела који је искључиво намењен наставницима и предавачима наставе програмирања, и другог стручног дела намењеног како наставницима, тако и ученицима.

Област рачунарства и информатике широко је развијена, и у највећем броју случајева, реч је о наставном предмету који је омиљен од стране ученика. Планови и програми се разликују од смера до смера, али већина програма има неке заједничке области. То су области које обухватају рад у неком од апликативних програма из пакета MS Office, или у неком од програма за обраду цртежа, слика. Рад у овим програмима ученицима је интересантан и омогућава слободу у изражавању идеја и велики степен креативности. Употреба Интернета, способност сурфовања мрежом, вештина је коју слободно можемо рећи поседује свака особа између 5 и 55 година, а остали можда у нешто мањем проценту, али и даље веома високом.

Међутим, свет рачунарства не чине само визуелни ефекти, слике и шарени менији. Прави дигитални свет, свет који се крије иза сваке визуално уређене странице, је свет прецизно дефинисаног кода. Код је писан у једном од програмских језика, док у самој бити функционисања система наилазимо на низове нула и јединица. Ма колико то чудно изгледало ови низови, односно програмски код, пружају много већу слободу у креативном изражавању, нуде обиље могућности надохват руке. Само треба пружити руке и дотаћи их.

Са методичке тачке гледишта први задатак наставника јесте омогућити ученику сусрет са правим, огољеним, дигиталним светом.

Тај сусрет мора бити реализован као сусрет будућих нераздвојних пријатеља, а не као увод у заморне часове програмирања. Уколико наставник постигне да ученик програмирање третира као уметност изражавања, вајања и обликовања сопствених структура, идеја, мисли, постигао је потпуни успех.

Приликом програмирања основу рада представља изградња алгоритма. Реч је о прецизно дефинисаном путу који води до решења проблема. У програмерском свету егзистира велики број програмских језика, сваки од њих има своје предности, али и своје мане. За сваки програмски језик алгоритам је исти. Известан број програмера заобилази тај први корак у раду – писање алгоритма. Међутим, то је дозвољено само прекаљеним програмерима који пишу кодове по алгоритму који само они виде. Почетницима у свету програмирања преко је потребан план рада како би били успешни. Најбољи програмски језик за учење програмирања је програмски језик Pascal. Реч је о програмском језику који је и креиран са намером да поједностави и приближи поступак записа кода, ученицима. Тек након савладаног програмирања у овом програмском језику, ученици би требало да одаберу неки од језика који имају озбиљније структуре, и у стању су да реше сложеније проблеме. То наравно, не значи да Pascal нема могућности за сложеније захвате. Pascal је веома логичан, лако разумљив и читљив. Када корисник савлада Pascal, оспособљен је за прелазак на виши ниво, да ли ће његов избор бити програм Delphi или C++, на кориснику је да одлучи. Delphi је програмски пакет који за кодирање користи објектни Pascal, тако да би овај избор представљао континуитет у раду. Савршен је за учење објектно оријентисаног програмирања. Када ученик добро савлада Delphi, прелазак на C++ није тежак, јер се разлике очитују само у резервисаним речима и нешто другачијој синтакси, чак штавише, корисник може програм писан у Delphi-ју ред по ред „преписати“ у C++.

Следи списак наставних тема са циљевима и исходима, за сваку понаособ. Наравно приручник не може у потпуности заменити класичну припрему за час, припрема је законска обавеза сваког предавача, али може у великој мери олакшати њено писање и бити њен саставни део. Треба имати у виду да приручник не прати стриктно План и програм наставе програмирања, будући да се планови и програми разликују од школе до школе, од смера до смера. Аутори су настојали

да глобално посматрају наставу програмирања, и да неке стандардне теме обухвате овим приручником, уз претпоставку да ће сваки наставник који буде користио овај приручник, додати своја опажања и елементе које жели да има у својој припреми за час.

I

Наставна тема	Решавање проблема помоћу рачунара.
Циљ	Стицање знања о начину решавања проблема алгоритамским путем.
Исход (стечена знања)	Ученик ће разумети поступак решавања проблема помоћу рачунара.

II

Наставна тема	Увод у програмски језик Pascal.
Циљ	Стицање знања о начину решавања проблема алгоритамским путем.
Исход (стечена знања)	Ученик ће разумети поступак решавања проблема помоћу рачунара.

III

Наставна тема	Елементи језика.
Циљ	Предочити ученику основне елементе програмског језика, значај и могућност употребе у свакодневном животу.
Исход (стечена знања)	Ученик ће знати да разликује типове података у Pascal-у и реши задатке једноставних линијских структура.

IV

Наставна тема	Одлуке.
Циљ	Стицање знања о начину решавања проблема употребом условних алгоритамских корака.
Исход (стечена знања)	Ученик ће бити у стању да састави програм који нуди више алтернатива.

V

Наставна тема	Наредбе за понављање (циклуси).
Циљ	Ученицима омогућити решавање проблема који захтевају кораке са великим бројем понављања. Разумевање појма циклуса, као базичног појма програмирања.
Исход (стечена знања)	Ученик ће разумети предност употребе циклуса у односу на класично решавање проблема линијским путем.

VI

Наставна тема	Низови.
Циљ	Омогућити ученицима да самостално саставе програм у

	којем се ради са великим бројем података истог типа.
Исход (стечена знања)	Ученик ће разумети разлику између елемента низа и податка који није елеменат низа.

VII

Наставна тема	Подпрограми.
Циљ	Стицање знања о коришћењу подпрограма у ситуацијама када то олакшава рад.
Исход (стечена знања)	Ученик ће бити у стању да препозна ситуацију у којој решење помоћу процедуре или функције скраћује поступак рада.

VIII

Наставна тема	Слогови.
Циљ	Разумевање појма слога у програмирању.
Исход (стечена знања)	Ученик ће разумети појам слога и бити у стању да креира програм у којем ће дефинисати и тип података слог.

IX

Наставна тема	Датотеке.
Циљ	Стицање знања о начину трајног меморисања података на неком спољном носиоцу информација.
Исход (стечена знања)	Ученик ће разумети потребу за креирањем датотеке, ако жели да његови подаци остану трајно сачувани.

1. Основни појмови

1.1. Хардвер и софтвер

Сваки рачунарски систем обухвата техничку и програмску реализацију. За техничку реализацију употребљава се израз **хардвер** (hardware), док се за програмску користи термин **софтвер** (software).

Хардвер се најчешће дефинише као скуп свих механичких, електричних и електронских уређаја којима се врши обрада података, у најкраћем - то је машина.

Софтвер је скуп програма, правила, и свих пратећих објашњења везаних за рад рачунара. Део софтвера може писати сам корисник рачунарског система, да би извршио одређен задатак који стоји пред њим, мада се већина корисника не бави програмирањем. Такви корисници користе рачунар као алат да би извршили задатак, за који је алгоритам већ познат, односно програм већ написан.

Да би претходни садржаји били још јаснији извршићемо једно поређење, упоредићемо рачунар са човеком, хардвер би представљало људско тело, док би софтвер била душа.

Скуп програма који се користе у неким одређеним применама називају се кориснички или **апликативни програми**, данас су изузетно много распрострањени. Незамислив је било какав озбиљнији рад на рачунару без употребе неког од апликативних софтверских пакета. Корисници ове програме најчешће добијају од произвођача рачунара или од неке специјализоване софтверске фирме. У таквим случајевима учешће у програмирању је ограничено, сведено на минимум. Међутим, постоји известан број случајева где кориснику не стоје на располагању готови апликативни програми. Тада је корисник приморан да сам пише програм или изнајми некога ко ће тај посао радити за њега. Те програме могуће је касније продати другим корисницима, са сличним потребама. С друге стране, постоји читав низ програма који пружају основни сервис апликативним програмима. Ти програми, укључујући језичке преводиоце, заједнички се означавају као **системски софтвер**. Један од најважнијих елемената системског софтвера је оперативни систем чији сервис нормално укључује управљање улазним и излазним уређајима рачунара, запамћеним подацима и обезбеђује приступ рачунару истовремено од стране неколико корисника.

Рачунарски систем може се посматрати као хијерархијска структура софтверских и хардверских компонената. На највишем нивоу налази се апликативни софтвер, који се наслања на системски софтвер испод себе. С друге стране, системски софтвер се наслања на процесор, меморију и улазно-излазне уређаје који чине хардвер рачунара. Хијерархијска структура се у суштини не

завршава на овом нивоу, пошто су саме хардверске компоненте конструисане од једноставнијих компонената.

Разлика између хардвера и софтвера није тако оштра као што се може претпоставити на основу хијерархијске структуре. Свака компонента рачунарског система извршава одређене функције. Неке од тих функција (превођење програма) се традиционално извршавају софтверски; док се друге (сабирање) традиционално извршавају хардверски. Неке функције могу понекад да се извршавају софтверски, а понекад хардверски. Пример је множење које представља функцију коју обезбеђују сви рачунари. У принципу, све функције рачунарског система могу се реализовати било софтверски, било хардверски. То значи да алгоритам који описује извршавање те функције може бити реализован било у облику програма или уграђен у физичке компоненте рачунара. У пракси избор између хардвера и софтвера зависи од предности која се даје односу брзина-цена. Комплексне функције, такве као што су превођење са високо програмског језика или пројектовање мостова, генерално се реализују софтверски, јер би цена реализације специјализованог хардвера за те функције била огромна. Једноставније функције или функције чије се извршавање захтева веома често, по правилу се извршавају хардверски, јер се жели постизање што веће брзине извршавања. На основу овога може се закључити да се граница између хардвера и софтвера у погледу имплементације појединих функција не може стриктно дефинисати и да ће се и у будућности мењати са технолошким напретком.

1.2. Програмски језици

Правилна употреба природних језика подразумева познавање како синтаксних правила тако и семантике. Да би се олакшала комуникација између људи ради реализовања одређених послова осмишљени су вештачки језици, у такве спадају Есперанто, језик аритметичких израза итд. Посебно место у овом скупу заузимају програмски језици, језици који су намењени комуницирању на релацији човек – рачунар.

Програмски језик, дакле представља вештачки језик који омогућује запис поступака за решавање задатака на рачунару, при чему за писање записа није потребно познавање техничких карактеристика рачунара. Састоји се од симбола груписаних у речи. Те речи се називају кључне или резервисане речи. Да би се програмски језици могли користити потребно је да постоји програм који преводи програм корисника са програмског на машински језик. Такав програм назива се преводилац. Преводилац је у стању да препозна синтаксне грешке[1], док семантичке[2] може да препозна само човек.

Програмске језике делимо у неколико категорија, и то :

[1] Синтакса, у граматици се дефинише као наука о граматичкој исправности језичких конструкција.

[2] Семантика – обухвата значења језичких конструкција.

решавање нумеричких проблема у области технике	језици вештачке интелигенције	језици за системско и конкурентно програмирање
Pascal	Lisp	Ada
Fortran (Formula Translation)	Prolog	Modula -2
Algol (Algorithmic Language)	SMALLTALK	Blicc
C	C (иако спада у прву групу заузима значајно место у алаткама за рад јер се лако интегрише у UNIX који постаје стандард за многеAI радне станице)	Occam
BASIC (Beginners All purpose Symbolic Instruction Code)		
COBOL (Common Bussines Oriented Language)		

Сваки почетник у програмирању, приликом избора програмског језика у којем ће радити, мора водити рачуна о следећим елементима:

- да има једноставан и прегледан код (начин записа података и наредби)
- да се често среће у окружењу, сарадња међу програмерима од великог је значаја,
- да подржава рад са графиком, игрице, рад у мрежи и слично.

Почетници често не могу да се одлуче између два или више програмских језика. Сви језици имају исти циљ – решење проблема, разлика је само у начину на који долазе до решења. Програмски језици се могу посматрати као различити дијалекти једног истог језика. Због тога је веома важан план решавања задатка – алгоритам. Исправан алгоритам налази примену у свим језицима.

1.2.1. Машински језик

Конструкција модерних рачунара омогућава смислено реаговање рачунара само на одређени тип импулса, реч је о бинарним електричним сигналима који чине машински језик. Инструкције и бројеви су кодирани помоћу различитих комбинација нула и јединица. У машинском језику свака програмска инструкција има као резултат извршавање само једне операције рачунара. Програм написан на машинском језику потпуно је разумљив за рачунар, а за корисника рачунара веома мало, односно нимало. Програмирање у машинском језику захтева добро познавање хардверских компоненти и посебну припрему за рад. Овај језик не спада у природне језике, и да би се премостила разлика долази до увођења програмских језика. Разумљивији су човеку од машинског, а истовремено и ближи машинском у односу на природне.

1.2.2. Програми преводиоци

Преводилац програме писане у неком програмском језику преводи у програме на машинском језику. Изворни програм је програм писан у програмском језику, а преведени програм у машински језик назива се извршни програм. Развојем програмских језика развио се и велики број програма преводилаца. Програми преводиоци се уносе преко оперативног система у меморију и заузимају одређени простор.

Компајлер све фазе превођења обавља редом и коначан резултат превођења је извршни програм. Уколико постоје грешке исправљање програма је дуготрајно јер се извештај добија после завршетка свих фаза. Процес превођења треба понављати више пута све док се не отклоне све евентуалне грешке.

Интерпретер је преводилац који изворни програм преводи у извршни у току рада инструкцију по инструкцију. Нема меморисаног преведеног програма. Сваком извршавању програма следи превођење, па је извршавање успорено. Предност интерпретирања је да се свака инструкција одмах анализира и исправљање грешака је брже и лакше. Интерпретер се најчешће примењује када се жели директна комуникација човека са рачунаром, погодан је за почетнике.

2. Програмски језик Pascal

Програмски језик Pascal развио је Niklaus Wirth, члан Међународне федерације за обраду информација (Federation of Information Processing (IFIP) Working Group 2.1.). Програмски језици који су били развијени у његово време нису имали могућности извршавања одређених функција, програми су били сложени и слабо разумљиви ширем кругу корисника. Предак Pascal програмског језика је језик Algol 60, у чијем развоју је учествовао и Niclaus Wirth. Такође су преузете и неке компоненте из језика Algol 68 и Algol – W. Прва верзија Pascal-а објављена је 1971. године, а прва ревизија оригиналне верзије 1973. године.

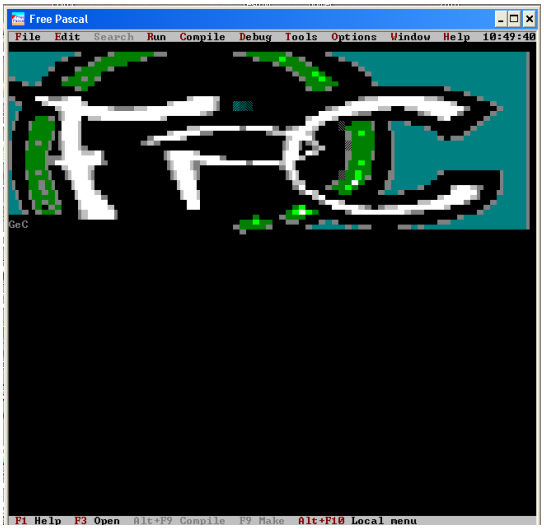
Језик је назван по великом француском математичару и филозофу Blaise Pascal-у (1623 – 1662.). Овај научник је одиграо значајну улогу у развоју светске науке. Истовремено Pascal се сматра и пиониром рачунарства. Конструисао је први механички калкулатор који је у основи имао могућност сабирања.

Овај програмски језик умногоме прати логику људског размишљања. Програмер своју идеју саопштава програмом, на тај начин, да наредбе прате мисаони процес човека. Pascal је један од првих структуралних језика, што подразумева креирање прегледне структуре програма и разлагање програма на мање делове од којих сваки представља засебну целину.

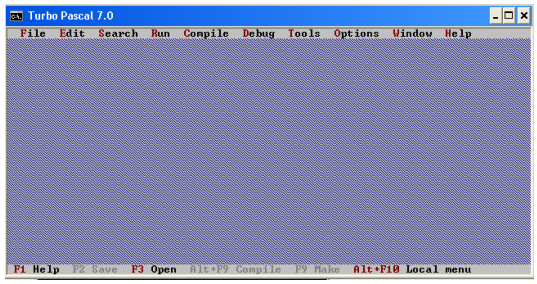
2.1. Pascal компајлери

Free Pascal

Free Pascal (FPK Pascal) је 32 и 64 битни Pascal компајлер. Подржан је на различитим процесорима: Intel x86, Amd64/x86_64, PowerPC итд. Што се тиче оперативног система такође подржава велики број: Linux, DOS, FreeBSD, Haiku, OS/2, Mac OS X/Darwin, Win32, Win64, WinCE, Netware. Последња верзија овог компајлера је верзија 2.4.0 објављена првог јануара 2010. године. Синтакса језика је потпуно усклађена са TP 7.0 (Turbo Pascal 7) и са већином верзија Delphi – ја.



Слика1. Радни екран Free Pascal-a



Слика2. Радни екран Turbo Pascal-a

Turbo Pascal

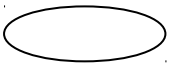
Реч је о софтверу који обухвата компајлер и интегрисано развојно окружење за програмски језик Pascal. Развила ја је фирма Borland, вођа пројекта био је Philippe Kahn. Прва верзија објављена је 1983. године, док се највише у употреби задржала верзија 7.0.

2.2. Алгоритми

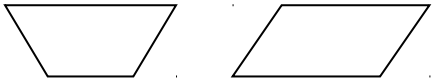
За историју рачунарства значајан је XI век, време у којем је живео и радио Muhammad ibn Musa Al'Khowarizmi (Abū Ja`farMuhammad ibn Mūsā al-Khwārizmī, ташкентски свештеник (Persian/Arabic) према неким изворима био је то седми и осми век). Развио је концепт писаног процеса који треба следити како би се постигао неки циљ. О својим истраживањима написао је књигу која је том концепту дала његово модерно име - алгоритам. При томе овај математичар је прецизно описао четири основне рачунске операције. Данас се развија посебна научна дисциплина која се бави теоријом алгоритама.

Не може се говорити о програмирању, а пренебрегнути појам алгоритам. Алгоритам представља основу у процесу програмирања. Алгоритхм- алгоритам, дефинише се као прописани низ тачно дефинисаних правила, шематски приказаних, или поступака за решавање неког проблема. Решавање се одвија у коначном броју корака.

Постоји велики број алгоритамских корака, најчешће су у употреби:

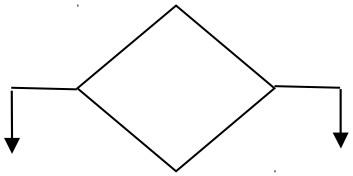


- старт или почетак алгоритма

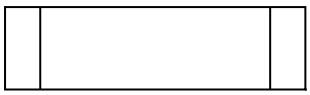


- улазне величине

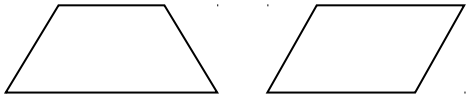
- обрада, аритметичке операције, доделе вредности



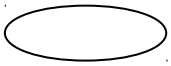
- условни корак, одлука



- набрајање, пребројавање, понављање



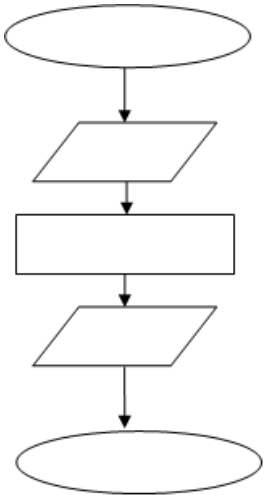
- излазне величине, резултати, поруке



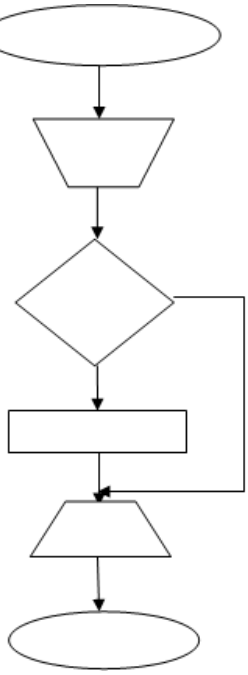
- крај алгоритма

Основна подела алгоритама је на:

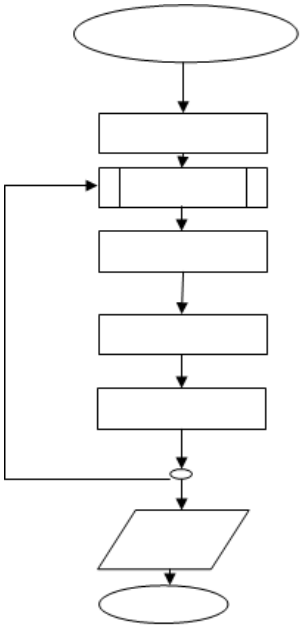
1. Линијске – сваки корак се извршава тачно једанпут, нема корака који се понављају и нема корака који се никада неће извршити.



Сл. 3 Линијски



Сл. 4 Разгранати

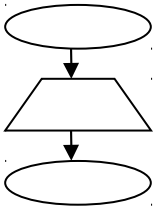


Сл. 5 Циклични

2. Разгранате – могу да постоје кораци који се никада неће извршити.
3. Цикличне – постоје кораци који се понављају одређени број пута.

2.3. Први Pascal Program

Pascal програм може бити веома једноставан. Следи пример програма који ће омогућити исписивање поруке на екрану "CAO KAKO SI"



```
{ -------------------------- Prvi program -------------------------- }
PROGRAM PrviProgram(OUTPUT);
BEGIN
 WRITELN('CAO KAKO SI') ; readln
END.
```

Без обзира да ли програм има велики или мали број наредби потребно је да поседује одређену структуру. Овај програм садржи наредбу улаза (WRITELN) којом се извршава исписивање поруке на екрану. Исписује се оно што је уписано између апострофа. Да би програм био извршен мора почети службеном речи BEGIN и завршити се службеном речи END. Ово је такозвано **тело програма** (или програмски блок), и обично садржи неколико логичких и аритметичких операција, а може садржати и рад са датотекама, речима. Службена реч BEGIN одговара почетном кораку старт у алгоритму.

Коментар:

{ -------------------------- Prvi program -------------------------- }

Овакав коментар компајлер ни не примећује, тј. он је невидљив у извршавању програма. Коментар се може наћи на било ком месту у Pascal

програму, а да нема никакав утицај на сам ток програма. Коментар се уписује између { и } или између (* и *).

На пример:

{ ovo je komentar} (* I ovo je komentar *)

Друга линија овог програма је такозвано **заглавље програма.** Заглавље садржи службену реч PROGRAM и име програма (PrviProgram), могуће је изабрати име програма које програмер жели, с тим да име не сме бити службена реч, или пак исто име као нека променљива у програму. Име програма такође не може да почиње цифром или специјалним знаком, али име може садржати цифру. Шематски основну структуру програма можемо представити на следећи начин :

Program ИмеПрограма;
- } одељак дефиниција и декларација { Const, Var, Type
-
-
Begin
- } одељак наредби { Read, Readln; If…then…; For…to…do While…do Repeat…until… Write, writeln
-
End.

Није обавезно поред имена програма, али у неким случајевима од велике је важности, писање службенех речи input и output. Оне представљају одговор на питање да ли програм има само излазне податке (OUTPUT), или и улазне и излазне (INPUT, OUTPUT), без ових службених речи програм може да ради.

Програмер сам бира да ли ће писати великим или малим словима, Pascal „не види“ да ли су слова велика или мала. Када чита променљиву A или a, за компајлер међу њима нема разлике. Дакле, програм је синтаксно добро записан у оба следећа случаја.

```
{ -------------------------- PRVI PROGRAM -------------------------- }
PROGRAM PRVIPROGRAM(OUTPUT);
BEGIN
 WRITELN('CAO KAKO SI') ; readln
END.
```

```
{ -------------------------- Prvi program -------------------------- }
Program PrviProgram(OUTPUT);
Begin
 Writeln ('CAO KAKO SI') ; readln
End.
```

Едитор Turbo Pascal- a

Постоје два начина за прављење програма у Pascal-у. Први се веома ретко користи, а реч је о употреби стандардног едитора, текст процесора веома слабих могућности. Други начин је употреба интегрисаног развојног окружења. Кориснику су на располагању едитор, компајлер, дебагер (проверава исправност програма, указује на грешке и отклања их).

2.3.1. Наредба излаза: WRITELN, WRITE

Ако програмер жели да одређени текст буде исписан у више редова, један испод другог користи команду **WRITELN** као у наредном примеру:

```
{ --------------------------- PROGRAM ZA PRIKAZ TEKSTA--------------------------- }
PROGRAM Jedno ispod drugog(OUTPUT);
BEGIN
 WRITELN('CAO.');
 WRITELN('Sta da radimo danas?');
 WRITELN('Da li da radimo Pascal?') ; readln
END.
```

Приликом извршавања програма резултат ће изгледати овако:

```
CAO.
Sta da radimo danas?
Da li da radimo Pascal?
```

Али ако се програмеру не допада приказ на екрану, лако ће променити излаз у, рецимо, следећи облик:

```
CAO. Sta da radimo danas?
Da li da radimo Pascal?
```

Решиће то лако кад се обрише једно LN, након ког неће доћи до пребацивања у нови ред.

```
{ --------------------------- PROGRAM ZA PRIKAZ TEKSTA 1--------------------------- }
PROGRAM Jedno ispod drugog(OUTPUT);
BEGIN
WRITE ('CAO.');
 WRITELN('Sta da radimo danas?');
 WRITELN('Da li da radimo Pascal?'); readln
END.
```

Једина разлика међу командама WRITE и WRITELN је у томе што ће се након команде WRITELN следећи запис појавити у реду испод.

2.3.2. Рад са бројевима

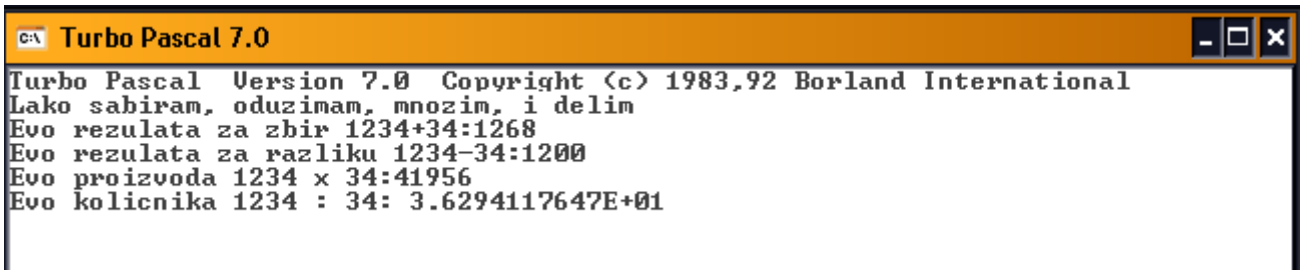
Основне операције:

	Сабирање	одузимање	множење	дељење
Математика	+	-	X	:
Pascal	+	-	*	/

У следећем програму постоји приказ за све аритметичке операције:

```
{ ------ PROGRAM ZA PRIKAZ ARITMETICKIH OPERACIJA 2 ------}
PROGRAM MatematickeOperacije(OUTPUT);
BEGIN
  WRITELN('Lako sabiram, oduzimam, mnozim, i delim');
  WRITELN('Evo rezulata za zbir 1234+34:', 1234+34);
  WRITELN('Evo rezulata za razliku 1234-34:', 1234-34);
  WRITELN('Evo proizvoda 1234 x 34:', 1234*34);
  WRITELN('Evo kolicnika 1234 : 34:', 1234/34);              readln
END.
```

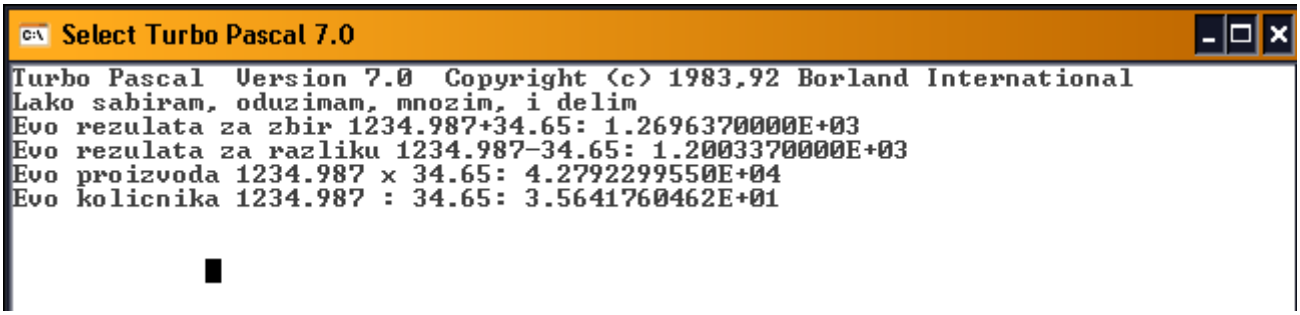
Након покретања овог програма на екрану ће се појавити:



Све лепо изгледа, али тамо где је дељење појављује се E+01, да ли је то могуће другачије приказати на излазу? Већ следећи пример има пуно више проблема са приказом података:

```
{ ------ PROGRAM ZA PRIKAZ ARITMETICKIH OPERACIJA 2 ------}
PROGRAM MatematickeOperacije(OUTPUT);
BEGIN
  WRITELN('Lako sabiram, oduzimam, mnozim, i delim');
  WRITELN('Evo rezulata za zbir 1234.987+34.65:', 1234.987+34.65);
  WRITELN('Evo rezulata za razliku 1234.987-34.65:', 1234.987-34.65);
  WRITELN('Evo proizvoda 1234.987 x 34.65:', 1234.987*34.65);
  WRITELN('Evo kolicnika 1234.987 : 34.65:', 1234.987/34.65);      readln
END.
```

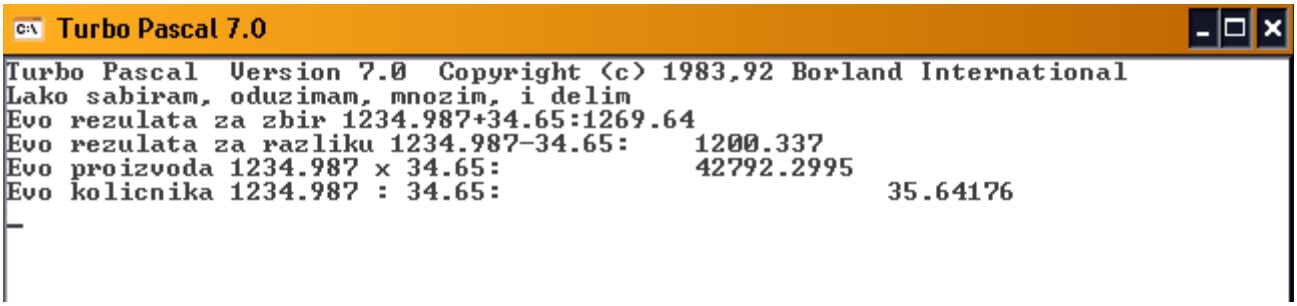
Након покретања овог програма на екрану ће се појавити:



Видљиво је да резултат није нарочито прегледан, са малим изменама у команди WRITELN корисник програма ће моћи да види адекватне резултате записане како треба, односно за око пуно лепше.

```
{ ------ PROGRAM ZA PRIKAZ ARITMETICKIH OPERACIJA 2 ------}
PROGRAM MatematickeOperacije(OUTPUT);
BEGIN
  WRITELN('Lako sabiram, oduzimam, mnozim, i delim');
  WRITELN('Evo rezulata za zbir 1234.987+34.65:', 1234.987+34.65:2:2);
  WRITELN('Evo rezulata za razliku 1234.987-34.65:', 1234.987-34.65:12:3);
  WRITELN('Evo proizvoda 1234.987 x 34.65:', 1234.987*34.65:22:4);
     WRITELN('Evo kolicnika 1234.987 : 34.65:', 1234.987/34.65:32:5);
   readln
END.
```

Након покретања овог програма на екрану ће се појавити:



Уочљиво је да се први број (:*2*:2, :*12*:3, :*22*:4, :*32*:5)“помера“ резултат у десно колико је потребно, односно како је програмер задао, а други број одређује број децималних места иза зареза (:2:*2*, :12:*3*, :22:*4*, :*32*:5). Након оваквих команди нема више записа као што је E+01 који представља број децималних места односно 10^1, на пример E+02 је 10^2, E+03 је 10^3, E-03 је 10^{-3}, и тако даље (овакав запис чита се „пута десет на“).

2.3.3. Цели и реални бројеви

Дељење које се представља уз помоћ знака / је такозвано реално дељење. Из претходних примера види се да је запис таквог дељења реалан број који је записан као:

5.0000000000E+02 што је идентично са математичким приказом 5 x 10^2,

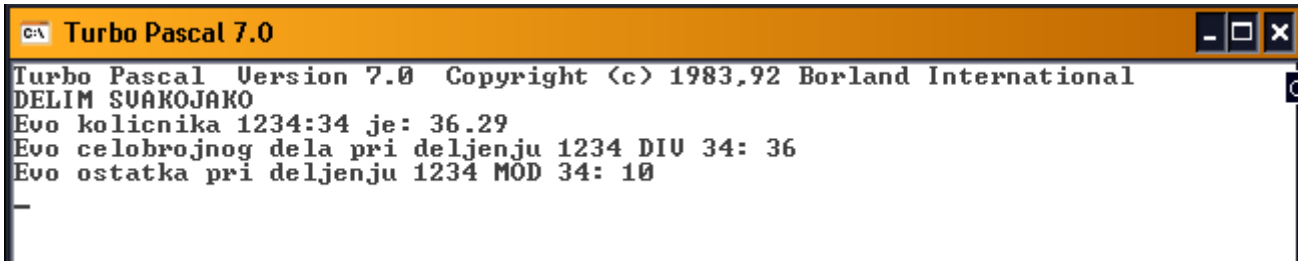
Ако је експонент иза Е негативан он представља додатни број децималних места, на пример:

5.525E-02 што је идентично са 0.05525, (потребно је само ограничити број децималних места).

Дакле, дељење које се представља са / је реално дељење јер је његов резултат увек реалан број. Поред овога у паскалу постоје и **DIV** и **MOD** који дају целобројна решења. У наредном примеру се види како то функционише:

```
{ ------ PROGRAM ZA PRIKAZ DELJENJA ------}
PROGRAM MatematickeOperacije(OUTPUT);
BEGIN
  WRITELN('DELIM SVAKOJAKO');
  WRITELN('Evo kolicnika 1234:34 je:', 1234/34:6:2);
  WRITELN('Evo celobrojnog dela pri deljenju 1234 DIV 34:', 1234  DIV 34:3);
    WRITELN('Evo ostatka pri deljenju 1234 MOD 34:', 1234 MOD 34:3);
   readln
END.
```

Након покретања овог програма на екрану ће се појавити:



Након дељења резултат је заузео 6 карактера и има две децимале као што је било задато.

Уз помоћ команде **DIV** добијен је цео део при дељењу два броја, док је децимални део једноставно одсечен.

Уз помоћ команде **MOD** добијен је остатак при дељењу два броја. И један и други резултат заузимају по три карактера као што је задато, па су за једно место одмакнути од текста.

Табела аритметичких операција				
ОПЕРАЦИЈА	**Аритметичка операција**	**Каква је операција**	**Какав је резултат када се изврши**	**Приоритет извршавања**
+	сабирање	Реална/целобројна REAL/INTEGER	Реалан/целобројни REAL/INTEGER	Низак Low
-	одузимање	Реална/целобројна REAL/INTEGER	Реалан/целобројни REAL/INTEGER	Низак Low
*	множење	Реална/целобројна REAL/INTEGER	Реалан/целобројни REAL/INTEGER	Висок High
/	реално	Реална/целобројна	Увек реалан број	Висок High

	дељење	REAL/INTEGER	REAL	
DIV	целобројно дељење	Целобројна INTEGER	Увек цео број INTEGER	Висок High
MOD	остатак при целобројном дељењу	Целобројна INTEGER	Увек цео број INTEGER	Висок High

2.4. Променљиве у Pascalu

Подаци се чувају у меморијским локацијама на специфичним адресама. Због тога одређене меморијске ћелије програмери заузимају променљивим. Ту уписују податке и читају их са истог места. Уколико се вредност променљиве промени уписује се на већ одређену локацију, тако да рачунар не губи време тражећи ново место за упис података.

2.4.1. Декларација променљивих

Ако се ради са променљивим вредностима користи се наредба за декларисање променљивих. У претходним примерима није било потребе за увођењем ове наредбе, јер се радило само са константним вредностима. Пре него што се променљиве користе у програму морају бити декларисане односно мора им се одредити тип. Тип променљиве се уписује у посебном делу програма који се зове део за декларацију (описивање променљивих). Овај део увек почиње наредбом VAR. Без обзира где се променљива први пут појављује, било то и на самом крају програма, мора бити декларисана – најављена у овој наредби.

```
VAR
    a: INTEGER;
    x: REAL;
```

Што значи да је a описана као цео број и може бити 2, 3, 4, 123,... а променљива x је реалан број и може бити и цео и реалан број, будући да је скуп целих бројева подскуп скупа реалних бројева, као на пример 2, 3, 4, 123, ... али и 2.5, 3.45, 1.23, 3.5E+02, ...

Ако жели да декларише више променљивих које су истог типа, тј исте врсте, програмер не мора сваку да описује посебно, већ може групу променљивих описати једном службеном речи:

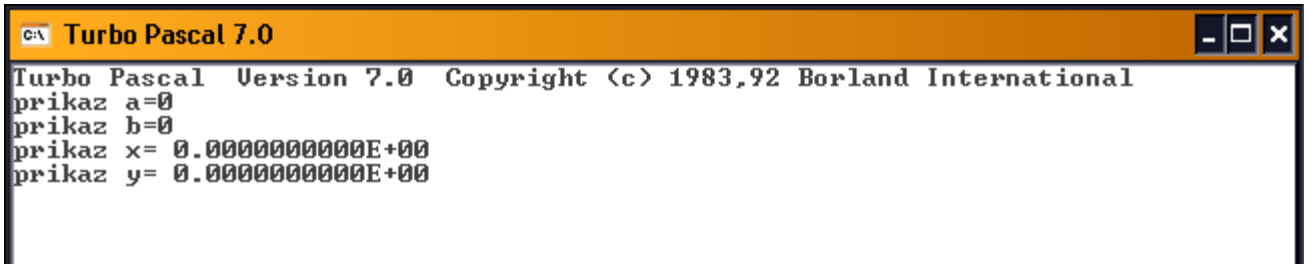
```
VAR
    a,b,c,d,e: INTEGER;
    x,y,k,m: REAL;
```

INTEGER И REAL се сврставају у стандардне типове података. Следећи програм показује како изгледа њихов запис на екрану након покретања програма.

```
{ -------------------------- prikaz REAL INTEGER -------------------------- }
```

```
PROGRAM Promrnljive (OUTPUT);
{ Deklaracija promenljivih }
VAR
  a,b :INTEGER;
  x,y :REAL;
{ Programski blok }
BEGIN
 WRITELN(‘prikaz a=',a);
 WRITELN('prikaz b=',b);
 WRITELN('prikaz x=',x);
WRITELN('prikaz y=',y); readln
END.
```

Екран након извршавања програма изгледа овако:



У претходном примеру промењљиве немају никакву вредност. Ако постоји потреба за додељивањем вредности промењљивој програмер користи знак „:=“ (узима вредност). Ова наредба (знак) често се назива и наредбом доделе. Примери:

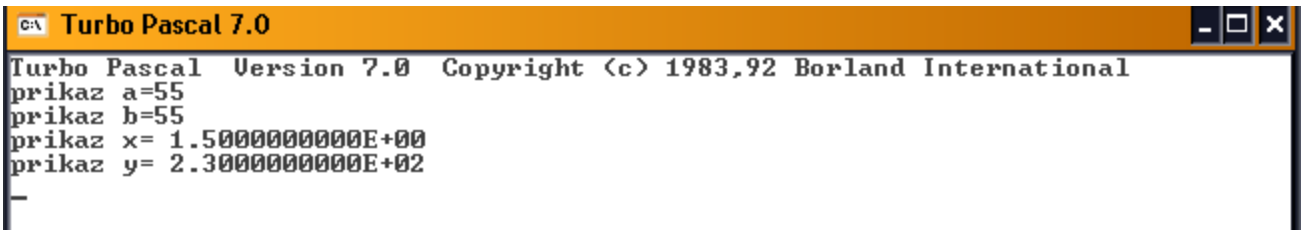
```
a:= 55;
b:= 55;
x:= 1.5;
y:= 2.3E+02;
```

Ако ове вредности убаци у претходни задатак програмер добија:

```
{ -------------------------- prikaz REAL INTEGER 1-------------------------- }
PROGRAM Promrnljive (OUTPUT);
{ Deklaracija promenljivih }
VAR
  a,b :INTEGER;
  x,y :REAL;
{ Programski blok }
BEGIN
{Nove ubacene promenljive}
a:= 55;
b:= 55;
x:= 1.5;
y:= 2.3E+02;
{program cita promenljive i ispisuje ih na ekran}
 WRITELN(‘prikaz a=',a);
 WRITELN('prikaz b=',b);
 WRITELN('prikaz x=',x);
WRITELN('prikaz y=',y); readln
```

END.

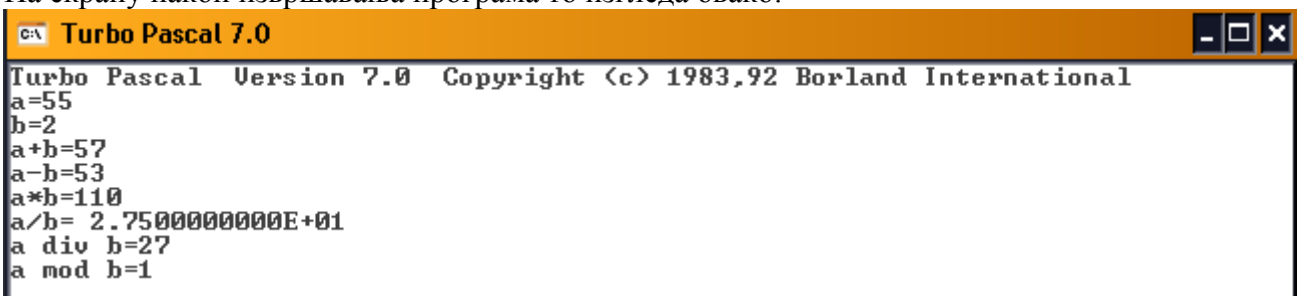
На екрану након извршавања програма то изгледа овако:



У наредном програму, две променљиве "a" и "b" су декларисане као целобројне. Уз помоћ наредбе излаза WRITELN програм ће приказати резултате разних аритметичких операција.

```
{ -------------------------- prikaz REAL INTEGER operacije -------------------------- }
        PROGRAM Operacije(OUTPUT);
        { deklarisanje promenljivih }
        VAR
         a, b :INTEGER;
        { Programski blok }
        BEGIN
         a:= 55;
         b:= 2;
         WRITELN('a=',a);
         WRITELN('b=',b);
         WRITELN('a+b=',a+b);
         WRITELN('a-b=',a-b);
         WRITELN('a*b=',a*b);
         WRITELN('a/b=',a/b);
         WRITELN('a div b=',a DIV b); { radi samo sa celim brojevima }
         WRITELN('a mod b=',a MOD b); { radi samo sa celim brojevima }
        readln
        END.
```

На екрану након извршавања програма то изгледа овако:



Као у математици и у програмирању је могуће да променљива узме вредност неке друге променљиве (на пример x:=y), или неке аритметичке операције са више променљивих (x:=(a+b)/2.31), али мора да се води рачуна о приоритету операција. Приоритет је исти као у математици. Дакле, ако пише a+b div 2 програм прво ради остатак при дељењу (b div 2) и тек онда сабира са a, ако пише (a+b) div 2 програм сабира (a+b) па тек онда проналази остатак при дељењу (a+b) div 2.

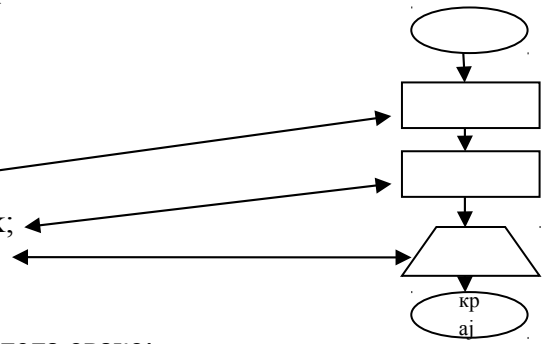
2.4.2. Дефинисање константи

У већини програмских језика могуће је дефинисати константе које се неће никада мењати при извршавању програма.

Као и променљиве и константе се декларишу на месту за декларацију (односно описивање), службена реч за описивање константи је CONST. Константе могу бити нумеричке или текстуалне. Употреба константи је од великог значаја у програмима који су гломазни па је отежано проналажење појединих променљивих, односно константи, а потребно је вршити ажурирање или промене појединих елемената. Такође и у програмима у којима се ради са веома великим бројевима.

Следећи пример показује како се константе користе у програму:

```
{ -------------------------- KONSTANTE -------------------------- }
PROGRAM Konstante(OUTPUT);
{ deklarisanje konstanti }

CONST
 Pi = 3.14159;
{ deklarisanje promenljivih }
VAR
 Poluprecnik, Obim :REAL;
{ Programski blok }
BEGIN
 Poluprecnik:= 4.9;
 Obim:= 2 * Pi * poluprecnik;
 WRITELN('Obim=', Obim)
END.
```



Када се изврши програм излаз изгледа овако:

2.4.3 ROUND, TRUNC

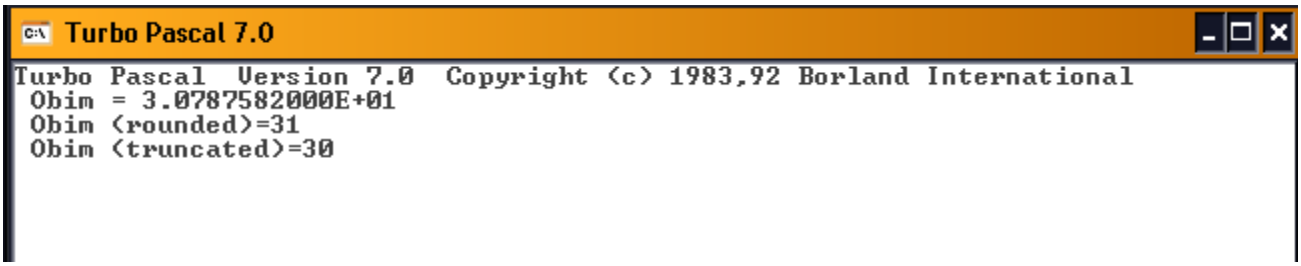
Ако програмер жели да ради уз помоћ целих бројева на располагању су му две операције које реалне бројеве претварају у целе.

ROUND(n)	Претвара реалан "n" у њему најближи цео број
TRUNC(n)	Одсеца од реалног броја "n" све што се налази иза зареза

Следећи пример показује како се **ROUND, TRUNC** користе у програму:

```
{ -------------------------- realan u ceo broj -------------------------- }
```

```
PROGRAM RealanUCeo(OUTPUT);
{ deklarisanje konstanti }
CONST
 Pi = 3.14159;
{ deklarisanje promenljivih }
VAR
 Obim, Poluprecnik :REAL;
 ZaokruzenObim, ObimBezZareza :INTEGER;
{ Programski blok }
BEGIN
 Poluprecnik:= 4.9;
 Obim:= 2*Pi* Poluprecnik;
 ZaokruzenObim:= ROUND(Obim);
 ObimBezZareza:= TRUNC(Obim);
 WRITELN(' Obim =', Obim);
 WRITELN(' Obim (rounded)=', ZaokruzenObim);
 WRITELN(' Obim (truncated)=', ObimBezZareza);readln
END.
```

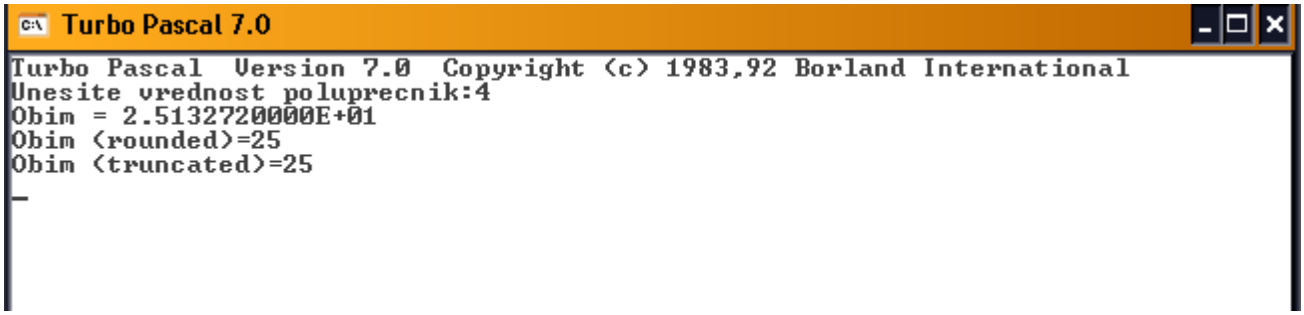


2.5. Учитавање са тастатуре, наредба улаза READLN, READ

Команде **READLN, READ** су наредбе улаза. Свака од њих зауставља извршење програма и очекује унос са тастатуре од стране корисника програма. Уз помоћ ових команди је могуће унети једну или више променљивих. Најбоље је користити наредбу у комбинацији са наредбом излаза како би корисник знао шта ради.

На следећем примеру се види како функционише програм када се користе ове наредбе:

```
{ -------------------------- NAREDBA ULAZA -------------------------- }
PROGRAM UnosPodataka(OUTPUT);
{ deklarisanje konstanti }
CONST
 Pi = 3.14159;
{ deklarisanje promenljivih }
VAR
 Obim, poluprecnik :REAL;
 ZaokruzenObim, ObimBezZareza:INTEGER;
{ Programski blok }
BEGIN
 WRITE('Unesite vrednost poluprecnik:');
 READLN(poluprecnik);
 Obim:= 2*Pi* poluprecnik;
 ZaokruzenObim:= ROUND(Obim);
 ObimBezZareza:= TRUNC(Obim);
 WRITELN('Obim =', Obim);
 WRITELN('Obim (rounded)=', ZaokruzenObim);
 WRITELN('Obim (truncated)=', ObimBezZareza)
END.
```

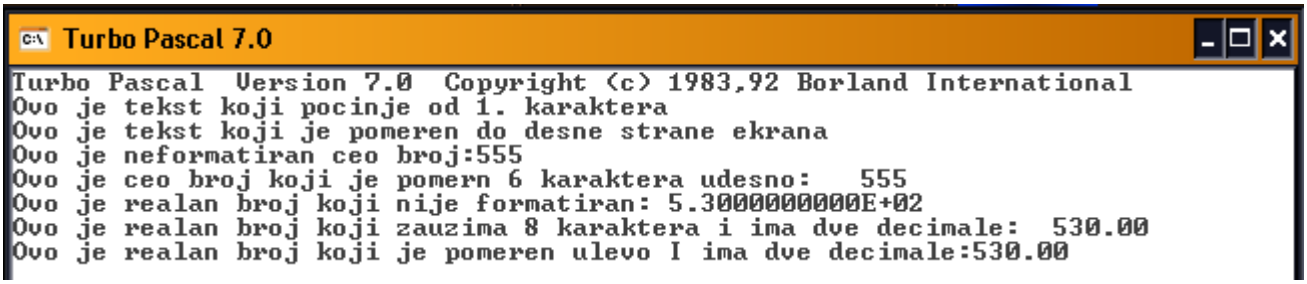


2.6. Форматирање ИЗЛАЗА

Међу првим примерима постоје задаци у којима је излаз форматиран, али следећи пример показује како је најбоље, тј најлакше направити излаз тако да буде прихватљивог изгледа.

```
{ -------------------------- FORMATIRANJE IZLAZA -------------------------- }
PROGRAM Format(OUTPUT);
{ deklarisanje promenljivih }
VAR
 a :INTEGER;
 b :REAL;
{ Programski blok }
BEGIN
a:= 555;
b:= 5.3e+02;
 WRITELN('Ovo je tekst koji pocinje od 1. karaktera');
 WRITELN('Ovo je tekst koji je pomeren do desne strane ekrana':50);
 WRITELN('Ovo je neformatiran ceo broj:', a);
 WRITELN('Ovo je ceo broj koji je pomern 6 karaktera udesno:', a:6);
 WRITELN('Ovo je realan broj koji nije formatiran:',b);
  WRITELN('Ovo je realan broj koji zauzima 8 karaktera i ima dve decimale:',b:8:2);
 WRITELN('Ovo je realan broj koji je pomeren ulevo I ima dve decimale:',b:0:2)
END.
```

Овако изгледа излаз након покретања програма:



3. Елементи језика

Стандардни типови података и стандардне функције

Подаци у сваком програмском језику могу бити цели или реални бројеви, стрингови за текст, или пак логички тип података, али сваки тип података мора бити декларисан на месту за декларацију, очему је већ било речи. Pascal поседује стандардне типове података из којих се изводе остали типови.

INTEGER	цели бројеви
REAL	реални бројеви
CHAR	карактери
BOOLEAN	логички тип може бити тачан или нетачан

У претходним примерима већ су кориштени INTEGER и REAL за декларисање нумеричких константи и промењљивих. Тестиране су и функције ROUND и TRUNC. А сада нешто више о могућим декларацијама константи и промењљивих.

3.1. Нумерички тип података

INTEGER

Turbo Pascal ЦЕЛОБРОЈНИ ТИП		
ТИП ПОДАТАКА	ВЕЛИЧИНА (у бајтима)	Област дефинисаности
SHORTINT	1	од -128 до +127
BYTE	1	од 0 до 255
INTEGER	2	од -32768 до +32767
WORD	2	од 0 до 65535
LONGINT	4	од -2,147,483,648 до +2,147,483,647

REAL

Turbo Pascal РЕАЛАН ТИП			
ТИП ПОДАТАКА	ВЕЛИЧИНА (у бајтима)	ПРЕЦИЗНОСТ	Област дефинисаности
SINGLE	4	7 цифара	од 0.71E-45 до 3.4E+38
REAL	6	11 цифара	од 2.94E-39 до 1.7E+38
DOUBLE	8	15 цифара	од 4.94E-324 до 1.79E+308
EXTENDED	10	19 цифара	од 3.3E-4932 до 1.18E+4932

COMP	8	Само цели бројеви	±9.2E+18

3.1.1. Стандардне аритметичке функције

Pascal има могућност да велике бројеве уз помоћ функција предефинише тако да могу да се користе у програму. Функције можемо поделити у три групе:

A. Функције претварања
B. Тригонометријске функције
C. Разноврсне функције

Стандардне аритметичке функције			
Изглед функције	**Резултат извршене функције**	**Тип параметра**	**Тип резултата**
Функције претварања:			
ROUND(x)	x се заокругљује на најближи цео	REAL	INTEGER
TRUNC(x)	x остаје без децималних места	REAL	INTEGER
*__Тригонометријске функције:__			
ARCTAN(x)	Аркустангес од x	REAL/INTEGER	REAL
COS(x)	Косинус од x	REAL/INTEGER	REAL
SIN(x)	Синус од x	REAL/INTEGER	REAL
Разноврсне функције:			
ABS(x)	Апсолутна вреднист од x	REAL/INTEGER	REAL/INTEGER
EXP(x)	Експненцијална функција од x (e^x)	REAL/INTEGER	REAL
LN(x)	Природни логаритам од x	REAL/INTEGER	REAL
SQR(x)	Квадрат од x (x^2)	REAL/INTEGER	REAL/INTEGER
SQRT(x)	Квадратни корен од x (x)	REAL/INTEGER	REAL

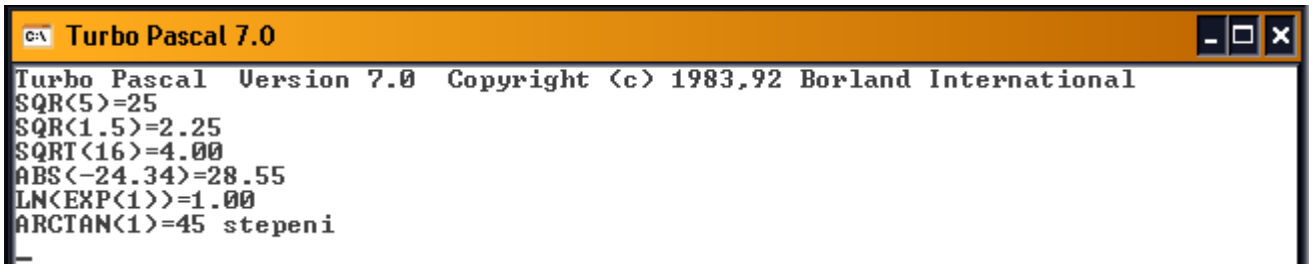
*__Сви углови морају бити у радијанима__

Излаз последње функције у наредном програму (ARCTAN) је конвертован у радијане. Треба обратити пажњу на који је начин програмер форматирао излаз.

```
{ ---------------------------- FUNKCIJE ------------------------------ }
{ Standardne aritmeticke funkcije }
PROGRAM funkcije (OUTPUT);
CONST
   Pi = 3.14159;{nije neophodno deklarisati konstntu, ali posto se vise puta ponavlja }
BEGIN
  WRITELN('SQR(5)=',SQR(5));
  WRITELN('SQR(1.5)=',SQR(1.5):0:2); { formatiranje izlaza }
  WRITELN('SQRT(16)=',SQRT(16):0:2);
  WRITELN('ABS(-24.34)=',ABS(-28.55):0:2);
  WRITELN('LN(EXP(1))=',LN(EXP(1)):0:2);
  WRITELN('ARCTAN(1)=',ARCTAN(1)* 180/Pi:0:0,' stepeni')
```

```
                { pretvaranje iz radijana u stepene }
        END.
```

Након покретања:



У следећем примеру се види како се врши степеновање у Pascal- у. Интересантно је напоменути да Pascal нема посебну функцију за извођење ове операције, за разлику од неких других програмских језика (Basic). Ако употреби знање које има о логаритамским и експоненцијалним функцијама програмер проблем степеновања може да реши овако:

```
{ ------------------------------ stepenovanje ------------------------------}
        { program stepenovanje }
        PROGRAM StepenBroja(INPUT,OUTPUT);
        VAR
         a, b:REAL;
        BEGIN
         WRITE('Unesi osnovu i stepen odvojeno praznim mestom:');
         READLN(a,b);
            WRITELN('Rezultat ',a:0:2,' podignuto na:',b:0:2,'stepen je: ',
EXP(LN(a)*b):0:2);readln
        END.
```

3.1.2. Turbo Pascal Адиционалне Функције

Од оваквих функција најчешће су у употреби:

FRAC(n**)**	Даје само разломљени део реалног броја "n"
INT(n**)**	Даје цео део реалног броја "n"

На пример:

```
WRITELN(FRAC(5.45):2:2);      даје 0.45
WRITELN(INT(5.45):2:2);  даје 5.00
```

Обе функције су реални бројеви!

Следеће функције генеришу случајне бројеве:

RANDOM Даје случајним избором цео број између 0 и целог броја "n" (нула је укључена).

RANDOM Даје случајним избором реалан број између 0 и 1 (нула је укључена).

3.2. Тип карактер: Char

CHAR тип се користи за резервисање једног карактера у Pascal програмском језику.

Сваки карактер може да с представи у једном бајту као цео број. Универзални код код персоналних рачунара је ASCII код (American Standard Code for Information Interchange). Он садржи 256 карактера од 0 до 255. Први део кода (од 0 до 127) је стандардан на свим персонални рачунарима:

- Велика слова енглеског алфабета (A-Z): ASCII 65 to 90
- Мала слова енглеског алфабета (a-z): ASCII 97 to 122
- Цифре (0-9): ASCII 48 to 57

Друга половина кода није стандардна и разликује се код различитих произвођача.

3.2.1. Стандардне функције за карактере

Ово су четири најчешће коришћене функције са карактерима:

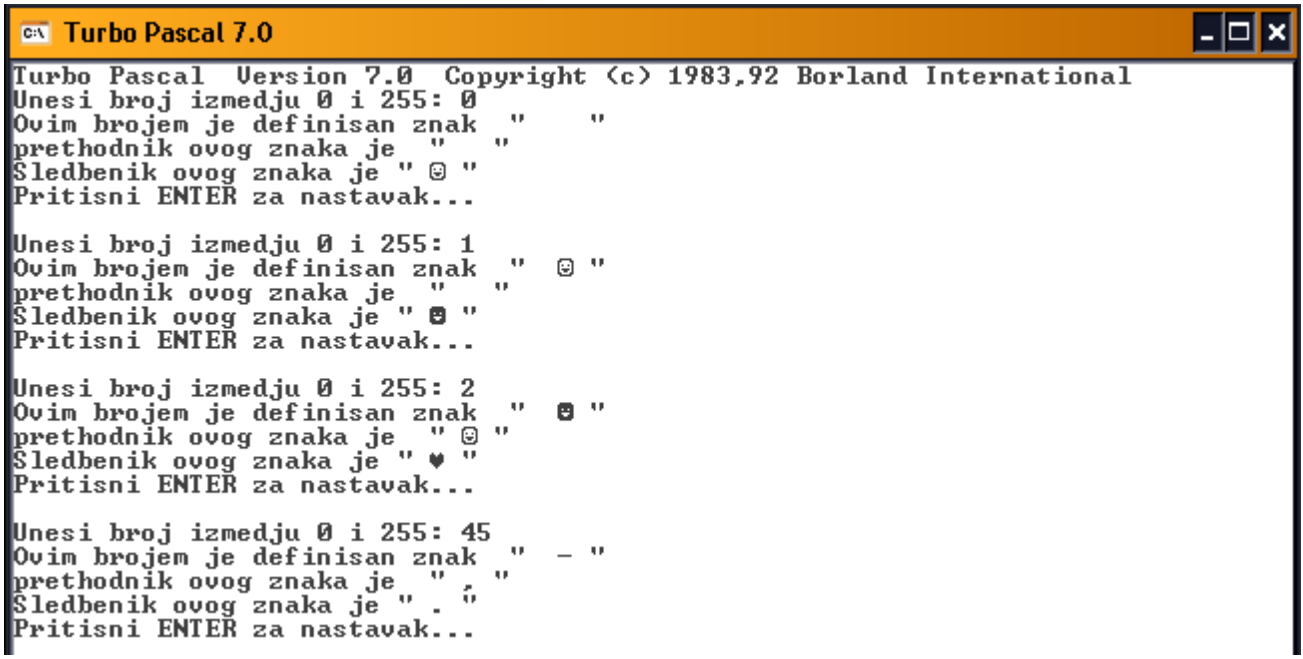
ORD(c) Претварање у број карактер "c"
CHR(n) Претварање у карактер броја "n"
PRED(c) Приказивање предходника каракеру "c"
SUCC(c) Приказивање следбеника карактера "c"

На следећем примеру се види како се користе претходне функције

```
{ ------------------------------ standardne f-je karaktera ------------------------------ }
{ pretvaranje, prethodnik, sledbenik }
PROGRAM PretPrdSled(INPUT,OUTPUT);
VAR
 Znak: CHAR;
Broj:BYTE;
BEGIN
WRITE('Unesi ceo broj izmedju 0 i 255: ');
 READLN(Broj);
 WRITELN('Ovim brojem je definisan znak  "  ', CHR(Broj),' " ');
Znak := CHR(broj);
 WRITELN('prethodnik ovog znaka je  " ',PRED(znak), ' " ');
 WRITELN('Sledbenik ovog znaka je " ',SUCC(znak),' " ');
 WRITELN('Pritisni ENTER za nastavak...');
 READLN
```

END.

Након покретања програма неколико пута резултати у зависности од улаза су:



3.2.2. String у стандардном Pascal језику

Декларација се у суштини не разликује много у стандардном и Турбо Pascal-у. Јер и један и други стринг виде као низ карактера. О низовим ће бити више речи у даљем тексту.

У стандардном Pascal -у декларација изгледа овако:

```
VAR
 ImePromenljive: PACKED ARRAY[1..15] OF CHAR;
```

Оваква декларација функционише и у Турбо Паскалу.

3.3 ТИП STRING

У Турбо Pascal -у може се дефинисати тип STRING на следећи начин, са обзиром да се налази у изведеним типовима података:

```
VAR
 ImeUcenika: STRING[20];
```

Број 20 означава број места која су резервисана за низ карактера. Корисник може унети текст са највише двадесет знакова.

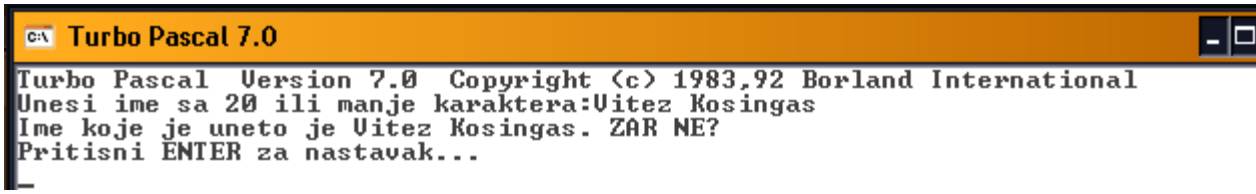
Уколико се број у загради не наведе сматра се да је подразумевана максимална вредност 255 знакова.

Тип стринг интересантан је када се ради са великим бројем текстуалних података над којима се врше неке операције, сортирање, претрага, брисање и слично.

Следећи пример показује начин рада са овим типом:

```
{ ------------------------------ TIP STRING ------------------------------ }
PROGRAM KakoString(INPUT,OUTPUT);
VAR
 Ime:STRING[20];
BEGIN
 WRITE('Unesi ime sa 20 ili manje karaktera:');
 READLN(Ime);
 WRITELN('Ime koje je uneto je ',Ime, '. ZAR NE?');
WRITELN('Pritisni ENTER za nastavak...');
 READLN
END.
```

Након покретања програма дешава се:



3.3.1. Дужина стринга

Наредба којом програмер може да види дужину стринга и покаже ја на екрану или искористи за даљу обраду података је LENGTH. Ако ову команду применимо у претходном програму то би изгледало овако:

```
{ ------------------------------ TIP STRING ------------------------------ }
PROGRAM KakoString(INPUT,OUTPUT);
VAR
 Ime:STRING[20];
BEGIN
 WRITE('Unesi ime sa 20 ili manje karaktera:');
 READLN(Ime);
 WRITELN('Ime koje je uneto je ',Ime, '. ZAR NE?');
 WRITELN('broj karaktera koji zauzima:',ime,' je:',LENGTH(Ime));
WRITELN('Pritisni ENTER za nastavak...');
 READLN
END.
```

Након покретања програма добијамо следећи резултат:

О стринговима ће више бити речи у даљем тексту.

3.4. ЛОГИЧКИ ТИП (BOOLEAN)

Логичке промењљиве могу да имају две вредности тачно (TRUE) или нетачно (FALSE). Назван је по Енглеском матаматичару Џорџу Булу (George Boole (1815-1864)).

Логичке операције

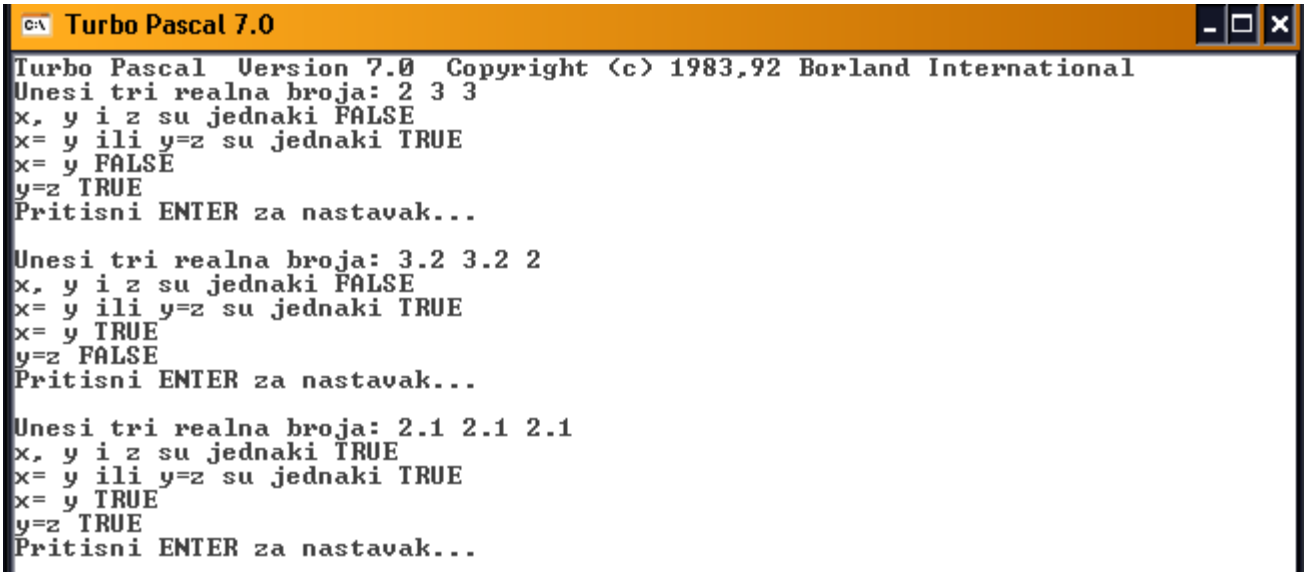
Оператор	Значење	Пример
>	веће од	A > B
<	мање од	C < 54
>=	веће или једнако	x >= 16.8
<=	мање или једнако	A+B <= 255
=	једнако	SQR(B) = 4*A*C
<>	различито	CHR(a) <> 'N'

3.4.1. Сложени логички изрази

Логичке операције у којима се користе само изрази из табеле су једноставни логички изрази, а ако се користе и „и“, „или“, и „не“ (AND, OR и NOT) у комбинацији са основним говоримо о сложеним логичким изразима.

```
{ ----------------------------- logicke operacije ----------------------------- }
{ uporedjivanje }
PROGRAM Uporedjivanje(INPUT,OUTPUT);
VAR
 x, y,z :REAL;
 Rezultat, Rezultat1, Rezultat2, Rezultat3:BOOLEAN;
BEGIN
 WRITE('Unesi tri realna broja: ');
 READLN(x, y, z);
Rezultat:= x=y ;
Rezultat1:= y=z ;
Rezultat2:= (x=y) and (y=z) ;
Rezultat3:= (x=y) or  (y=z) ;
WRITELN('x, y i z su jednaki ', rezultat2);
WRITELN('x= y ili y=z ', rezultat3);
WRITELN('x= y ', rezultat);
WRITELN('y=z ', rezultat1);
 WRITELN('Pritisni ENTER za nastavak...');
 READLN
END.
```

Након неколико покретања када се програм изврши добија се:

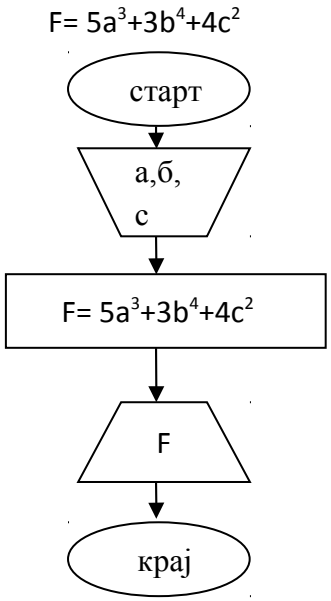


Приоритет операција у програмском језику Pascal

ОПЕРАЦИЈА	***ПРИОРИТЕТ***
NOT	Приоритет 1 (highest)
* / DIV MOD AND	Приоритет 2
+ - OR (XOR in Turbo Pascal)	Приоритет 3
= > < >= <= <>	Приоритет 4 (lowest)

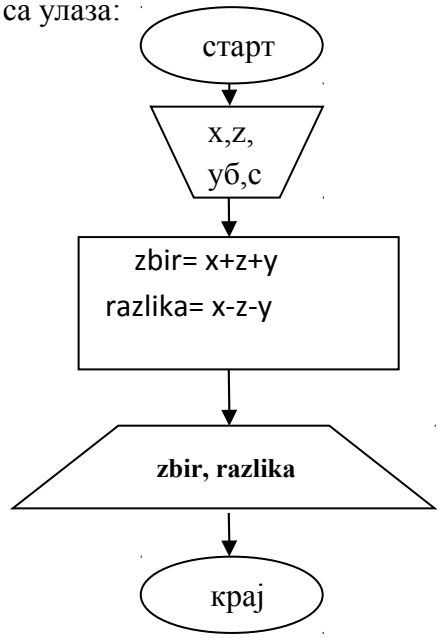
Задаци линијских алгоритамских структура

1. Саставити алгоритам и програм за израчунавање :




```
Program Z1;
Var a,b,c,F: real;
Begin
Writeln('Unesi tri broja');
Readln(a,b,c);
F:=5*a*a*a+3*sqr(b)*sqr(b)+4*sqr(c);
Writeln('F=', F);
Readln;
End.
```

2. Саставити алгоритам и програм за израчунавање збира и разлике три броја са улаза:




```
Program Z2;
Var x,z,y,zbir,razlika: real;
Begin
Writeln('Unesi tri broja');
Readln(x,y,z);
zbir:=x+z+y;
razlika:=x-y-z;
Writeln('zbir=', zbir);
Writeln('razlika=', razlika);
Readln;
End.
```

4. Одлуке

4.1. Доношење одлука

У свакодневном животу се често доносе одлуке типа ако - онда. Када нека особа прелази улицу најпре погледа семафор и размишља на следећи начин:

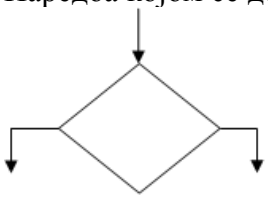
„**АКО** је зелено **ОНДА** прелазим, **У СУПРОТНОМ** чекам зелено“.

Начин функционисања наредбе за доношење одлуке приказаћемо на следећем примеру:

Особа која није пунолетна не може да купи алкохол. Када би то било написано у Pascal програмском језику изгледало би овако:

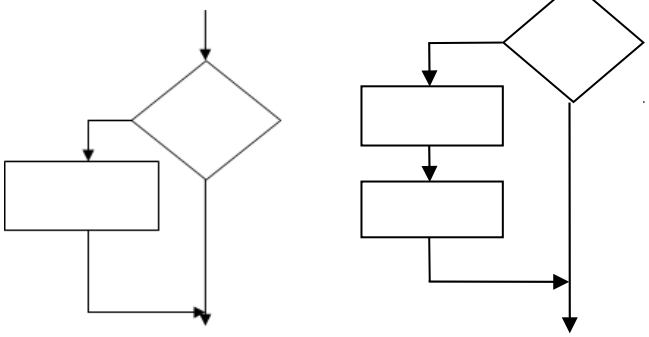
If Godine < 18 THEN WRITELN (‘Zalim, alkohol nije za maloletnike’);

Наредба којом се доносе одлуке алгоритамски се приказује:



Општи облици наредбе гласе:

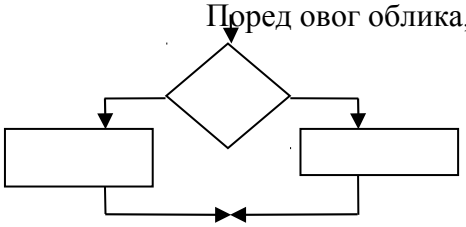
If uslov **then** naredba**;** **If** uslov **then begin** naredba1, naredba2**;end;**



Ако је услов задовољен извршава се наредба, у супротном, прелази се на следећи корак. Не постоји алтернатива.

Ако је услов задовољен, а потребно је извршити више наредби, обавезно се наредбе убацују између службених речи begin и end. Уколико корисник заборави да упише ове речи, компајлер ће извршити само прву наредбу која следи непосредно после одлуке, остале ће занемарити. Треба напоменути да уметање службених речи begin и end не омета извршавање ако после одлуке следи само једна наредба.

Поред овог облика, постоји и облик наредбе одлуке са обе алтернативе:



i**f** uslov **then** naredba1 **else** naredba2;

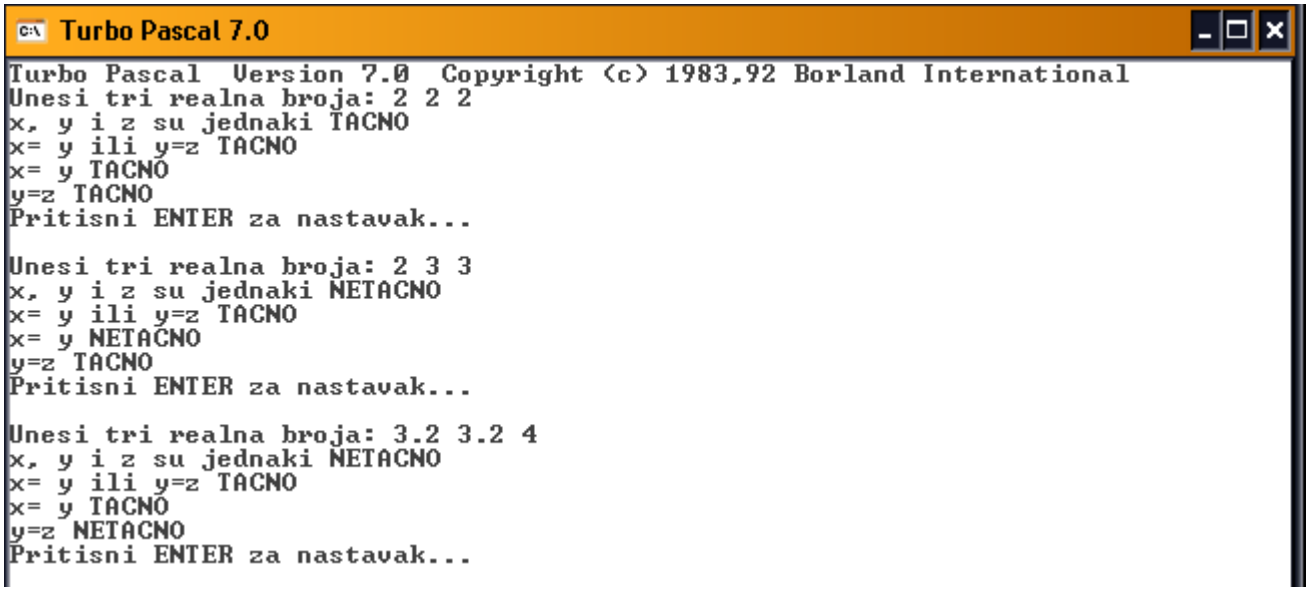
Ако је услов задовољен извшава се наредба1, у супротном наредба2.

И у овом случају, потребно је уметнути службене речи begin и end, ако се после одлуке извршава више од једне наредбе.

Задатак који је решен у делу Сложени логички изрази, сада ће се решити помоћу наредбе за одлучивање:

```
{ ----------------------------- upotreba IF THEN ----------------------------- }
{ uporedjivanje }
PROGRAM Uporedjivanje(INPUT,OUTPUT);
VAR
 x, y, z :REAL;
 Rezultat, Rezultat1, Rezultat2, Rezultat3:BOOLEAN;
Tekst, Tekst1, Tekst2, Tekst3:string;
BEGIN
 WRITE('Unesi tri realna broja: ');
 READLN(x, y, z);
   Rezultat:= x=y ;
              Rezultat1:= y=z ;
                        Rezultat2:= (x=y) and (y=z) ;
                                  Rezultat3:= (x=y) or  (y=z) ;
   If  rezultat = TRUE then Tekst := 'TACNO';
   If  rezultat = FALSE then Tekst:='NETACNO';
              If  rezultat1 = TRUE then Tekst1 := 'TACNO';
              If  rezultat1 = FALSE then Tekst1:='NETACNO';
                        If  rezultat2 = TRUE then Tekst2 := 'TACNO';
                        If  rezultat2 = FALSE then Tekst2:='NETACNO';
                                  If  rezultat3 = TRUE then Tekst3 := 'TACNO';
                                  If rezultat3 = FALSE then Tekst3:='NETACNO';
                        WRITELN('x, y i z su jednaki ', tekst2);
                                  WRITELN('x= y ili y=z ', tekst3);
   WRITELN('x= y ', tekst);
              WRITELN('y=z ', tekst1);
 WRITELN('Pritisni ENTER za nastavak...');
 READLN
END.
```

Када се програм стартује за различите вредности резултати су следећи:



Ако у претходни пример додамо одлуку У СУПРОТНОМ програм ће бити краћи и изгледати овако:

```
{ ------------------------------ upotreba IF THEN ------------------------------ }
{ uporedjivanje }
PROGRAM Uporedjivanje(INPUT,OUTPUT);
VAR
 x, y,z :REAL;
 Rezultat, Rezultat1, Rezultat2, Rezultat3:BOOLEAN;
Tekst , Tekst1, Tekst2, Tekst3:string;
BEGIN
 WRITE('Unesi tri realna broja: ');
 READLN(x, y, z);
   Rezultat:= x=y ;
            Rezultat1:= y=z ;
                     Rezultat2:= (x=y) and (y=z) ;
                              Rezultat3:= (x=y) or  (y=z) ;
   If  rezultat = TRUE then Tekst := 'TACNO' else  Tekst:='NETACNO';
            If      rezultat1   =   TRUE   then   Tekst1   :=   'TACNO'   else
Tekst1:='NETACNO';
                     If    rezultat2   =   TRUE   then   Tekst2   :=   'TACNO'   else
Tekst2:='NETACNO';
                              If  rezultat3 = TRUE then Tekst3 := 'TACNO' else
Tekst3:='NETACNO';
                     WRITELN('x, y i z su jednaki ', tekst2);
                              WRITELN('x= y ili y=z ', tekst3);
   WRITELN('x= y ', tekst);
            WRITELN('y=z ', tekst1);
 WRITELN('Pritisni ENTER za nastavak...');
 READLN
END.
```

Резултат покретања програма је исти као и у претходном примеру, али је програм прегледнији и краћи.

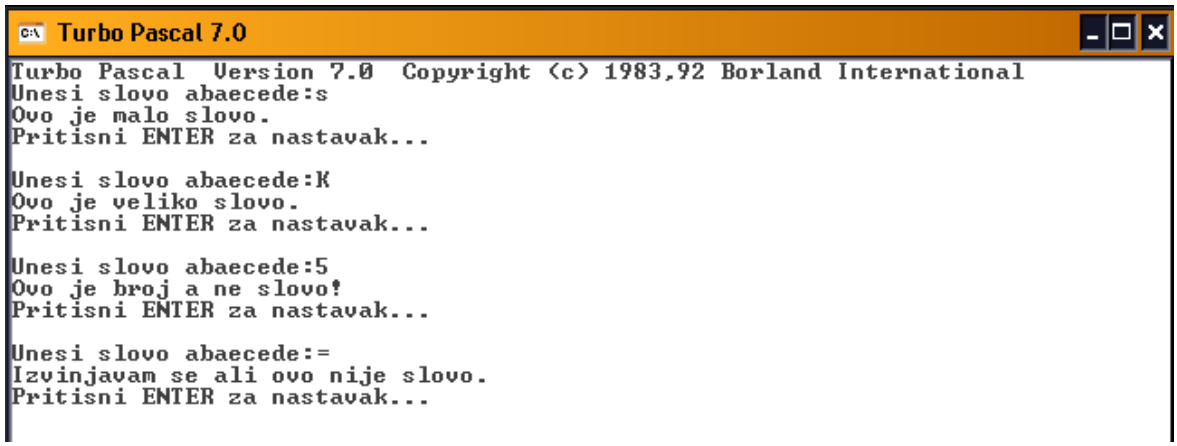
У претходном примеру се види како функционише IF-THEN-ELSE, али комбинује ли се више питања добија се нови начин питања и могућност решавања више проблема уз помоћ ELSE-IF

Логичке операције коришћене у програму користе ASCII код који за улазне карактере користи ORD функцију. Карактери се представљају у ASCII коду на следећи начин:

- Велика слова се представљају бројевима од 65 до 90.
- Мала слова се представљају бројевима 97 до 122.
- Цифре се представљају бројевима 48 до 57.

```
{ ------------------------------ upotreba ELSE-IF ------------------------------ }
PROGRAM Testerkaraktera(INPUT,OUTPUT);
VAR
 Slovo:CHAR;
BEGIN
 WRITE('Unesi slovo abaecede:');
 READLN(Slovo);
{ Pocetak IF naredbe }
{ ---------------------------- }
 IF (ORD(Slovo) > 64) AND (ORD(Slovo) < 91) THEN
   WRITELN('Ovo je veliko slovo.')
 ELSE IF (ORD(Slovo) > 96) AND (ORD(Slovo) < 123)THEN
   WRITELN('Ovo je malo slovo.')
 ELSE IF (ORD(Slovo) > 47) AND (ORD(Slovo) < 58) THEN
   WRITELN('Ovo je broj a ne slovo!')
 ELSE
   WRITELN('Izvinjavam se ali ovo nije slovo.');
{ Kraj pitanja }
{ ---------------------- }
WRITELN('Pritisni ENTER za nastavak...');
 READLN
END.
```

Када покрене програм корисник добија на екрану следеће одговоре:



4.1.1. Уметнути Услов

Ако је неопходно поставити више питања, тада програмер мора да зна када ће која операција бити извршена. Дакле када је један **IF** у оквиру другог операције се извршавају овим редом:

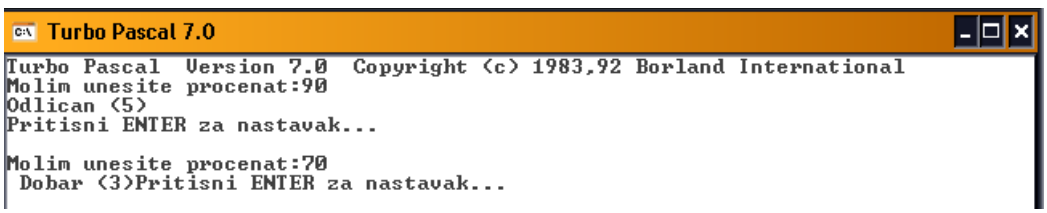
```
IF услов-1 THEN
    IF услов-2 THEN
        ...
        IF услов-n THEN
          операција-n1
        ELSE
          операција-n2
        ...
    ELSE
     операција-2
ELSE
 операција-1;
```

На конкретном примеру, то би изгледало овако:

Овај програм може да помогне наставнику око оцењивања ако су критеријуми који су задати следећи:

1. Оцена "5" проценат урађеног теста од 90% до 100%.
2. Оцена "4" проценат урађеног теста од 80% до 89%.
3. Оцена "3" проценат урађеног теста од 70% до 79%.
4. Оцена "2" проценат урађеног теста од 60% до 69%.
5. Оцена "1" проценат урађеног теста испод 60%.

```
{ ------------------------------ umetnuti uslov ------------------------------ }
PROGRAM Ocene (INPUT,OUTPUT);
VAR
Procenat:INTEGER;
BEGIN
 WRITE('Molim unesite procenat:');
 READLN(Procenat);
IF Procenat > 59 THEN                  { pocetak  IF naredbi }
  IF Procenat > 69 THEN
   IF Procenat > 79 THEN
     IF Procenat > 89 THEN
       WRITELN('Odlican (5)')
     ELSE
       WRITE('Vrlodobar (4)')
   ELSE
     WRITE(' Dobar (3)')
  ELSE
   WRITE(' Dovoljan  (2)')
 ELSE
  WRITE(' Nedovoljan (1)');   { Kraj IF naredbi }
WRITELN('Pritisni ENTER za nastavak...');READLN
 END.
```



Програм са истим резултатима се може решити и уз помоћ логичких израза и одлука на следећи начин:

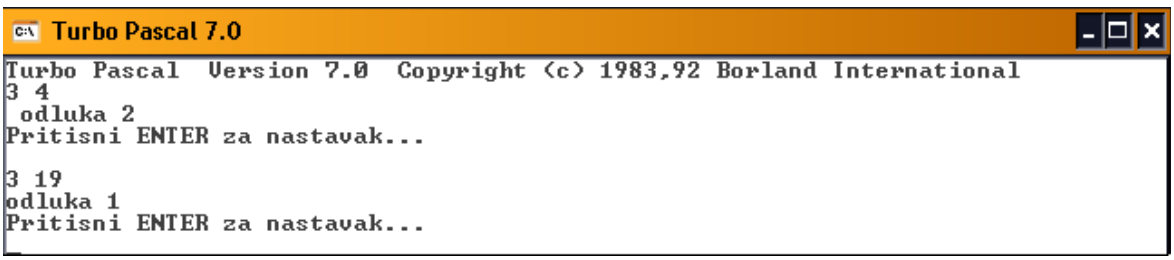
```
{ ---------- uslov + logicke ---------- }
PROGRAM Ocene1(INPUT,OUTPUT);
VAR
 Procenat       :INTEGER;
 A, B, C, D, F:BOOLEAN;
BEGIN
WRITE('Molim unesite procenat:');
 READLN(Procenat);
 A:= (Procenat >= 90) AND (Procenat <= 100);
 B:= (Procenat >= 80) AND (Procenat < 90);
 C:= (Procenat >= 70) AND (Procenat < 80);
 D:= (Procenat >= 60) AND (Procenat < 70);
 F:= (Procenat < 60) AND (Procenat >= 0);
IF A THEN                       { pocetak  IF naredbi }
  WRITELN ('Odlican (5)')
 ELSE IF B THEN
  WRITELN ('Vrlodobar (4)')
 ELSE IF C THEN
  WRITELN(' Dobar (3)')
 ELSE IF D THEN
      WRITE(' Dovoljan  (2)')
 ELSE IF F THEN
  WRITE(' Nedovoljan (1)')
 ELSE
  WRITELN('NEISPRAVAN UNOS!!!.');        { Kraj IF naredbi }
 WRITELN('Pritisni ENTER za nastavak...'); READLN
 END.
```

```
IF X >= 1 THEN
                BEGIN
                  IF Y >= 18 THEN
                WRITELN('odluka 1')
                  END
                ELSE
                WRITELN(' odluka 2 ');
```

Проверити да ли и овако постављено питање даје жељени резултат!!!

Наредни пример приказује како функционише претходна команда, ако се користи више команди одједном:

```
Program Објашњење: (INPUT,OUTPUT);
VAR X,y:real;
Begin
Readln(x,y);
IF X >= 1 THEN
   IF y >= 18 THEN
   WRITELN('odluka 1')
   ELSE
   WRITELN(' odluka 2 ');
WRITELN('Pritisni ENTER za nastavak...'); READLN
 END.
```



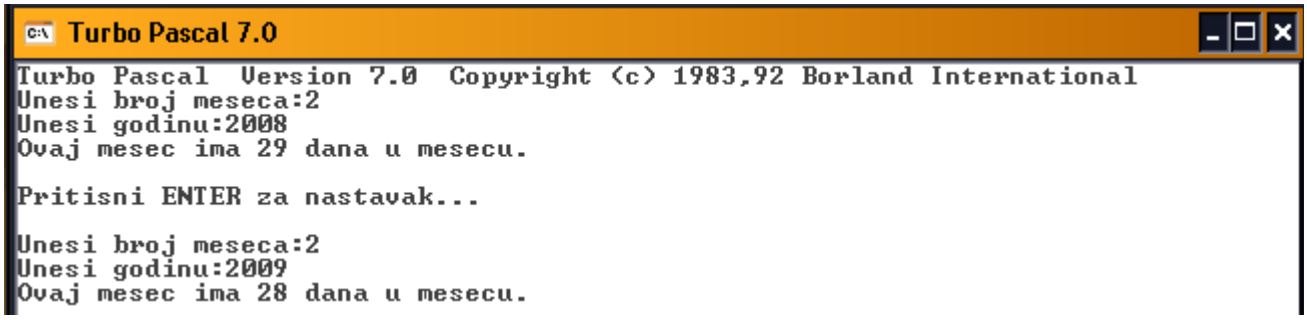
4.2. Вишеструки избор: CASE

CASE конструкција се користи када је потребно више алтернатива, као на пример у падајућем менију било ког апликативног софтвера. Форма ове наредбе изгледа овако:

```
CASE   PROMENLJIVA   OF
       oznaka-1 : operacija-1;
       oznaka-2 : operacija -2;
       ...
       oznaka-n : operacija -n;
   END
```

Пример: Број дана у месецу

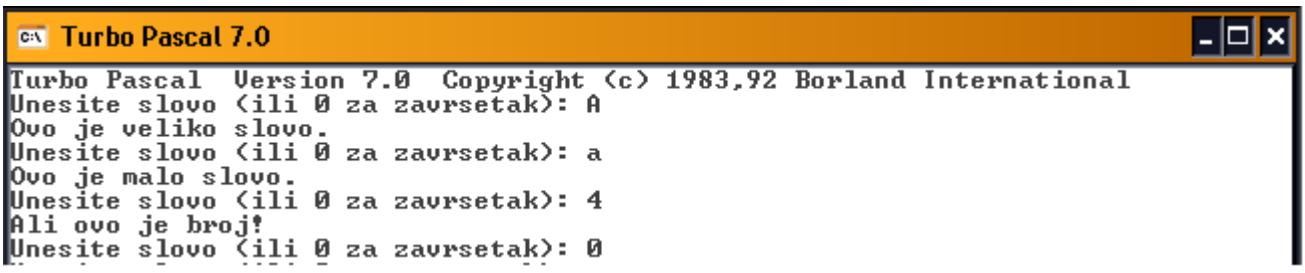
```
{ ----------------------------- Primer CASE ----------------------------- }
PROGRAM DaniuMesecu(INPUT,OUTPUT);
VAR
 Dani, Mesec, Godina:INTEGER;
BEGIN
 WRITE('Unesi broj meseca:');
 READLN(Mesec);
 CASE Mesec OF
  1,3,5,7,8,10,12 : Dani:= 31;
  4,6,9,11     : Dani:= 30;
  2          : BEGIN
               WRITE('Unesi godinu:');
               READLN(Godina);
               IF Godina MOD 4 = 0 THEN
                 Dani:=29
               ELSE
                 Dani:=28
              END;
 END;
  WRITELN('Ovaj mesec ima ',Dani,' dana u mesecu.');
WRITELN('Pritisni ENTER za nastavak...'); READLN
 END.
```



4.3. Безусловни прелазак: GOTO

Исказ **GOTO** се користи за пребацивање са једне линије програма на потребну линију. Ово је наредба за безусловни прелазак. Да би се користила неопходно је дефинисати ознаку или више њих (**label**) којима ће бити обележен ред на који треба да се пређе. Пример који следи показује на који се начин користи:

```
{ ------------------- Testiranje karaktera ------------ }
PROGRAM CharsTester2(INPUT,OUTPUT);
LABEL
 1, bb; { deklaracija label-a }
VAR
 UlazniKarakter:CHAR;
BEGIN
1:
 WRITE('Unesite slovo (ili 0 za zavrsetak): ');
 READLN(UlazniKarakter);
{ Pocetak IF-a }
IF UlazniKarakter = '0' THEN { uslov za izlaz }
   GOTO bb
 ELSE IF (ORD(UlazniKarakter) > 64) AND (ORD(UlazniKarakter) < 91) THEN
   WRITELN('Ovo je veliko slovo.')
  ELSE IF (ORD(UlazniKarakter) > 96) AND (ORD(UlazniKarakter) < 123)
THEN
   WRITELN('Ovo je malo slovo.')
 ELSE IF (ORD(UlazniKarakter) > 47) AND (ORD(UlazniKarakter) < 58) THEN
   WRITELN('Ali ovo je broj!')
 ELSE
   WRITELN(' OVO NIJE SLOVO !!! ');
{ Kraj IF-a }

 GOTO 1; { povratak na pocetak programa }
bb: { napustanje programa }
END.
```



4.4. Turbo Pascal Karakteristike: EXIT, CASE-ELSE

Следећи задатак показује на који начин ради комбинаација наведених наредби:

```
{ ------------------- DANI U NEDELJI ------------------- }
PROGRAM Dani(INPUT,OUTPUT);
Label Pocetak;
VAR
Dan:Integer;
BEGIN
Pocetak:
Writeln('Unesi redni broj dana u nedelji: '); Readln(Dan);
Case Dan of
1:writeln ('ponedeljak');
2:writeln ('Utorak');
3:writeln ('Sreda');
4:writeln ('cetvrtak');
5:writeln ('Petak');
6:writeln ('Subota');
7:writeln ('Nedelja');
Else exit {svi ostali slucajevi}
End; goto Pocetak;
 WRITELN('Pritisni ENTER za nastavak...'); READLN
End.
```

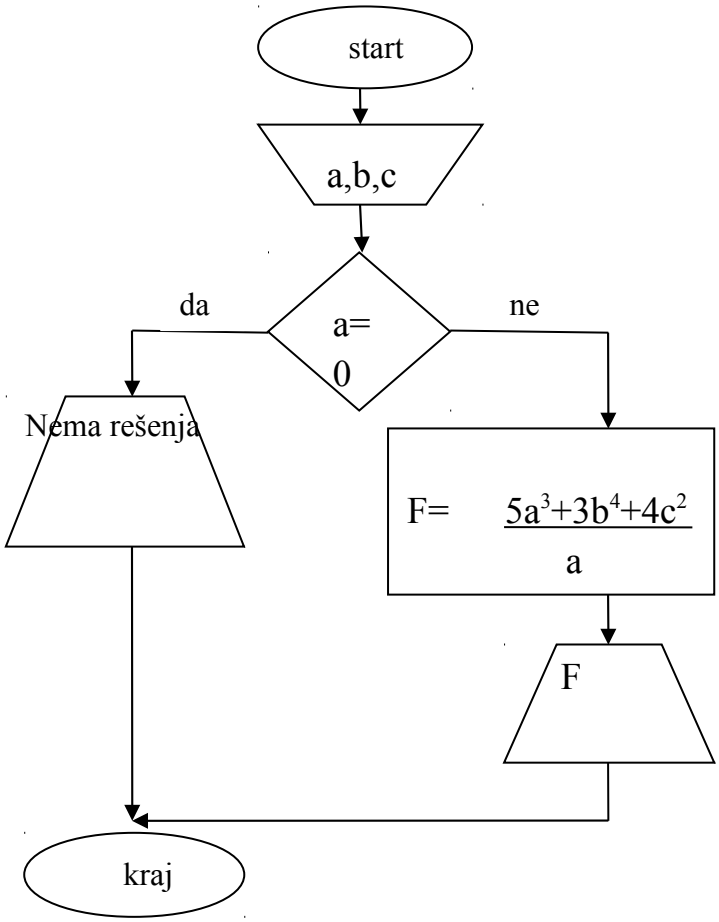
Дакле за бројне вредности од 1-7 приказује се одговарајући дан у недељи, ако није ни један од ових седам бројева програм се прекида.

Задаци разгранатих алгоритамских структура

1. Саставити алгоритам и програм за израчунавање :

$$F = \frac{5a^3+3b^4+4c^2}{a}$$

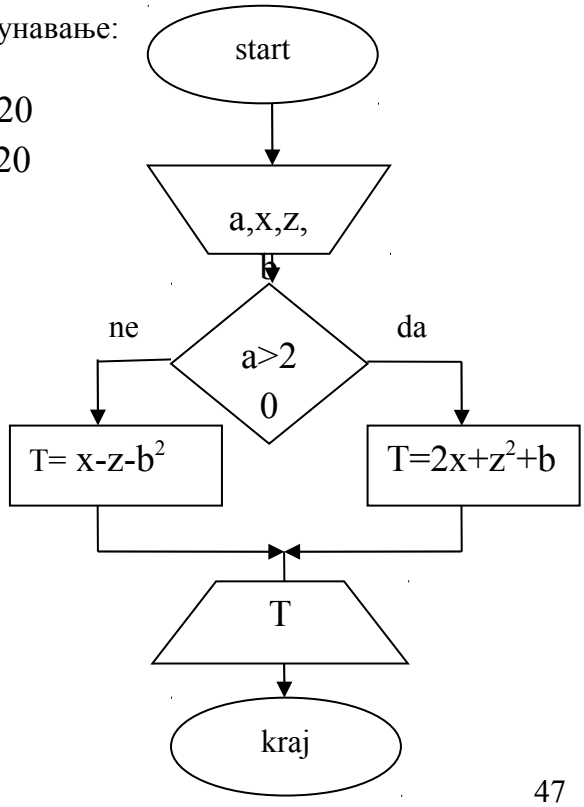
```
program z1;
var F,a,b,c:real;
begin
write('Unesi tri broja');
readln(a,b,c);
if a=0 then
write('Nema rešenja,
deljenje nulom nije definisano');
else begin
F:=(5*a*a*a+3*b*b*b*b+4*sqr(c))/a;
writeln('F=', F);
end;
end.
```



2. Саставити алгоритам и програм за израчунавање:

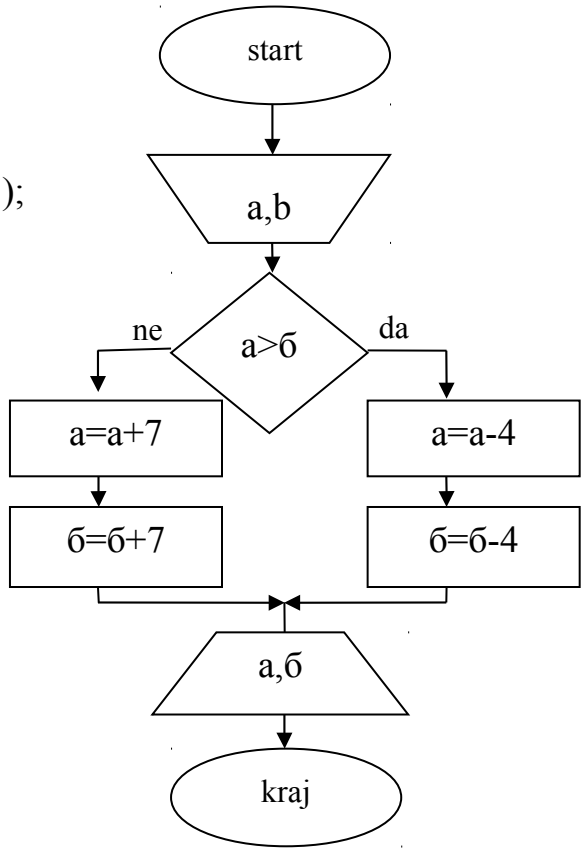
$$T = \begin{cases} 2x+z^2+b, & a>20 \\ x-z-b^2, & a\leq 20 \end{cases}$$

```
Program z2;
Var T,a,x,z,b:real;
Begin
Writeln('Unesi tri broja');
Readln(a,x,y,b);
If a>20 then T:=2*x+sqr(z)+b
Else T:=x-z-sqr(b);
Writeln('T=',T);
End.
```



3. Уносе се два броја, ако је први већи од другог сваки од бројева умањити за 4, у супротном сваки од бројева увећати за 7.

```
program z3;
var a,b:integer;
begin
writeln('Unesi dva cela broja');
readln(a,b);
if a>b then begin
a:=a-4;
b:=b-4;
end
else begin
a=a+7;
b=b+7;
end;
end.
```



Интересантно је напоменути да је у програмирању могуће новонасталу вредност назвати истим именом. Програм знак „:=„ види као доделу вредности – променљива а постаје нови број, памти се само нова вредност.

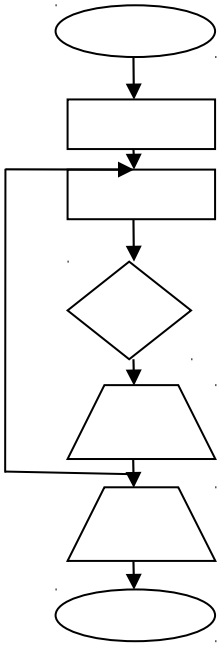
Пре службене речи ELSE не сме се писати знак „ ; “, ово је логичан след али се често деси грешка приликом писања. Свака смислена реченица – наредба у Pascal –у завршава се овим знаком, али ако реченица није завршена програм не дозвољава упис знака (не пише се после begin, do и слично).

5. Циклуси (петље)

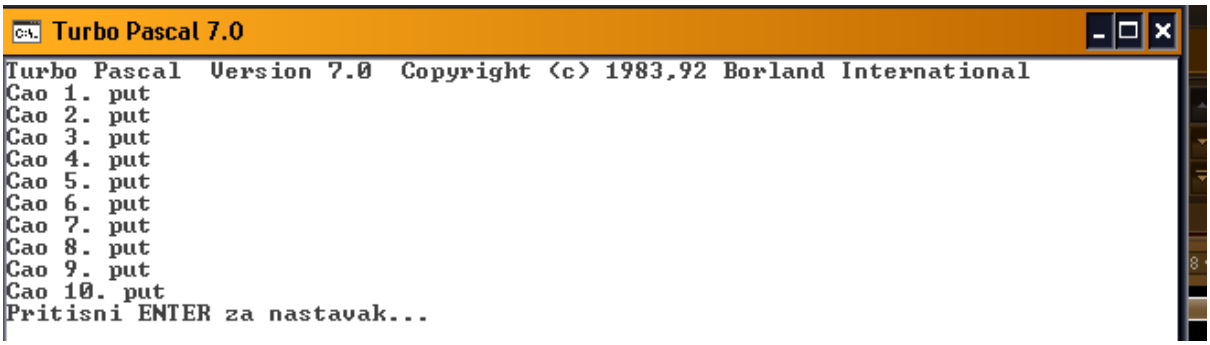
5.1 Циклус GOTO

У претходном поглављу је приказан начин функционисања наредбе GOTO. Уз помоћ ове наредбе је могуће направити бројач који би неколико пута на екрану исписао одређену поруку.

У наредном примеру се види како то изгледа:



```
{ ------------------ ciklus GOTO -------------------- }
PROGRAM GoToCiklus(OUTPUT);
LABEL
 Pocetak; { label declaration }
VAR
 Brojac :INTEGER;
BEGIN
 Brojac := 0;              {brojac se postavlja na 0}
Pocetak:                   {pocetak ciklusa}
 Brojac := Brojac + 1;     {brojac se uvecava za 1}
 IF Brojac <= 10 THEN
   BEGIN
    WRITELN('Cao ',Brojac, '. put');
   GOTO Pocetak          {povratak na pocetak ciklusa}
   END;
WRITELN('Pritisni ENTER za nastavak...'); READLN
END.
```



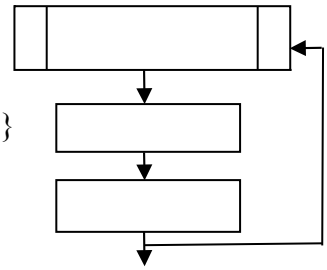
Наредба GOTO се у Pascal програмском језику ретко користи, али се кроз овај пример види како функционише циклус.

У овом програмском језику се користе следеће наредбе циклуса или петље:

- FOR цилкус
- WHILE циклус
- REPEAT циклус

5.2 FOR циклус

Уз помоћ овог циклуса је претходни задатак могуће урадити на пуно лакши начин пошто се након петље понавља једна или више команди:

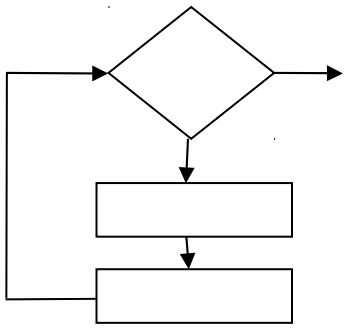


```
FOR brojac := 1 to 10 do {brojac uzima vrednost od 1 do 10}

FOR brojac := 10 downto 1 do {brojac uzima vrednost od 10 do 1}

{ ------------------ ciklus FOR---------------------- }
PROGRAM ForCiklus(OUTPUT);
VAR
 Brojac :INTEGER;
BEGIN
 For  Brojac := 1 to 10 do      {brojac uzima vrednost od 1 do 10}
   BEGIN
       WRITELN('Cao ',Brojac, '. put');  {moguce je ubaciti vise komandi izmedju
begin i end}
   END;
WRITELN('Pritisni ENTER za nastavak...'); READLN
END.
```

Излаз је исти као у претходном примеру.

5.3. WHILE циклус

Задатак који смо решавали уз помоћ претходног циклуса могуће је урадити и уз помоћ овог. Српски речено овај циклус би звучао овако док год важи услов понављај команду, или команде које се налазе између begin и end.



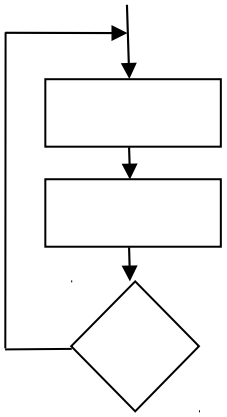
```
{ ------------------ ciklus WHILE ---------------------- }
PROGRAM WHILECiklus(OUTPUT);
VAR
 Brojac :INTEGER;
BEGIN
 Brojac := 1;                  {brojac se postavlja na 1}
While Brojac <= 10 do      {dok god vazi uslov radi}
    BEGIN
  WRITELN('Cao ',Brojac, '. put');
  Brojac := Brojac + 1;        {brojac se uvecava za 1}
    END;
WRITELN('Pritisni ENTER za nastavak...'); READLN
END.
```

Излаз је исти као у претходна два примера, пошто се решава исти проблем.

5.4. REPEAT циклус

Овај циклус би се на српски могао превести понављај команде док се не испуни услов.

```
{ ------------------ ciklus REPEAT -------------------- }
PROGRAM REPEATCiklus(OUTPUT);
VAR
 Brojac :INTEGER;
BEGIN
 Brojac := 0;                              {brojac se postavlja na 0}
REPEAT                    {ponavljaj}
    Brojac := Brojac + 1;                   {brojac se uvecava za 1}
    WRITELN('Cao ',Brojac, '. put');
UNTIL Brojac = 10;       {uslov je ispunjen}
WRITELN('Pritisni ENTER za nastavak...'); READLN
END.
```



Излаз је нормално идентитачан као у претходна три примера.

5.5. Циклус у цуклусу

У овом програмском језику могуће је формирати циклус унутар циклуса користећи све претходне, или исту врсту циклуса унутар циклуса. Ова могућност позната је и под именом концентрични циклус.

На пример рачунање факторијела за неки дати број све док се не стигне до јединице. Из математике нам је познато да се факторијел рачуна на следећи начин:

N!=N*(N-1)*...*1

```
{ ---------------------- faktorijel WHILE REPEAT--------------------------- }
PROGRAM faktorielWHILEREPEAT(INPUT,OUTPUT);
VAR
 Faktoriel   :REAL;
 Brojac, Broj :INTEGER;
BEGIN
 WRITE('Unesi neki broj (ili 0 za kraj): ');
 READLN(Broj);
 WHILE Broj <> 0 DO          { Pocetak WHILE ciklusa }
  BEGIN
   Faktoriel:= 1;
   Brojac:= 0;
   REPEAT                                          { pocetak REPEAT ciklusa }
     Brojac:= Brojac + 1;
    Faktoriel:= Faktoriel * Brojac;
    WRITELN('Faktoriel broja ', Brojac,' je ', Faktoriel:0:0);
   UNTIL Brojac = Broj;                            { kraj REPEAT ciklusa }
WRITE('Unesi neki broj (ili 0 za kraj): ');
   READLN(Broj)
  END;                        { Kraj WHILE ciklusa }
 WRITELN('Kraj!')
```

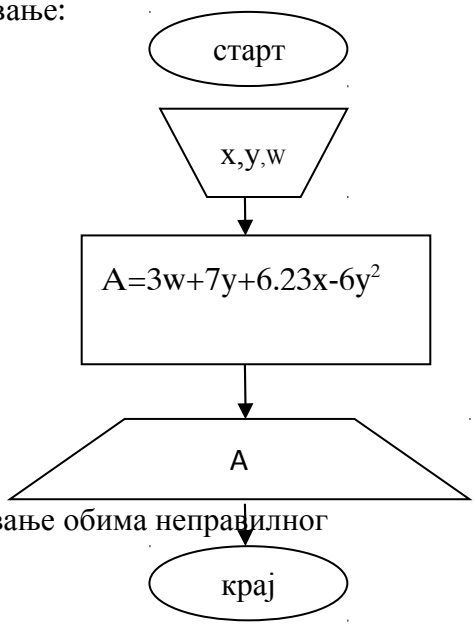
END.


```
{ ---------------------- faktorijel WHILE FOR--------------------------- }
PROGRAM faktorielWHILEFOR(INPUT,OUTPUT);
VAR
 Faktoriel   :REAL;
 Brojac, Broj :INTEGER;
BEGIN
 WRITE('Unesi neki broj (ili 0 za kraj): ');
 READLN(Broj);
 WHILE Broj <> 0 DO          { Pocetak WHILE ciklusa }
  BEGIN
   Faktoriel:= 1;
   FOR Brojac := 1 to Broj                    { pocetak FOR ciklusa }
     Begin
    Faktoriel:= Faktoriel * Brojac;
    WRITELN('Faktoriel broja ', Brojac,' je ', Faktoriel:0:0);
      End;                                    { kraj FOR ciklusa }
WRITE('Unesi neki broj (ili 0 za kraj): ');
   READLN(Broj)
  END;                         { Kraj WHILE ciklusa }
 WRITELN('Kraj!');
END.

{ ---------------------- faktorijel REPEAT FOR--------------------------- }
PROGRAM faktorielWHILEFOR(INPUT,OUTPUT);
VAR
 Faktoriel   :REAL;
 Brojac, Broj :INTEGER;
BEGIN
 WRITE('Unesi neki broj (ili 0 za kraj): ');
 READLN(Broj);
          REPEAT                              { Pocetak REPEAT ciklusa }
  BEGIN
   Faktoriel:= 1;
   FOR Brojac := 1 to Broj             { pocetak FOR ciklusa }
     Begin
    Faktoriel:= Faktoriel * Brojac;
    WRITELN('Faktoriel broja ', Brojac,' je ', Faktoriel:0:0);
      End;                               { kraj FOR ciklusa }
WRITE('Unesi neki broj (ili 0 za kraj): ');
   READLN(Broj)
          UNTIL Broj=0;                       { Kraj REPEAT ciklusa }
 WRITELN('Kraj!');
END.
```

Дакле исти проблем је решен на три различита начина, а исход је исти.

Задаци – пример писменог задатка

1. а) Написати алгоритам и програм за израчунавање:
   $A=3w+7y+6.23x-2,56y^2$



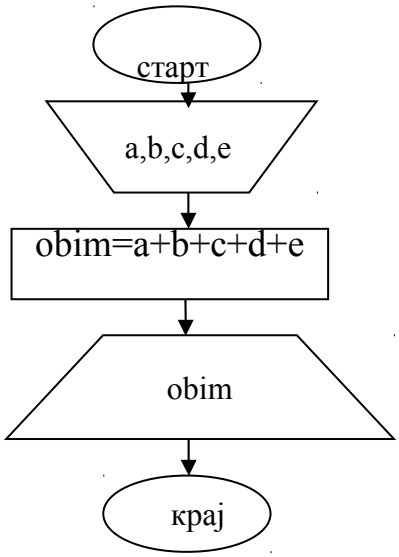
Решење:

```
Program z1;
Var A,w,y,x: real;
Begin
Read (w,x,y);
A:=3*w+7*y+6.23*x-6*sqr(y);
Write(A);
End.
```

b) Саставити алгоритам и програм за озрачунавање обима неправилног петоугла, ако су познате све странице.

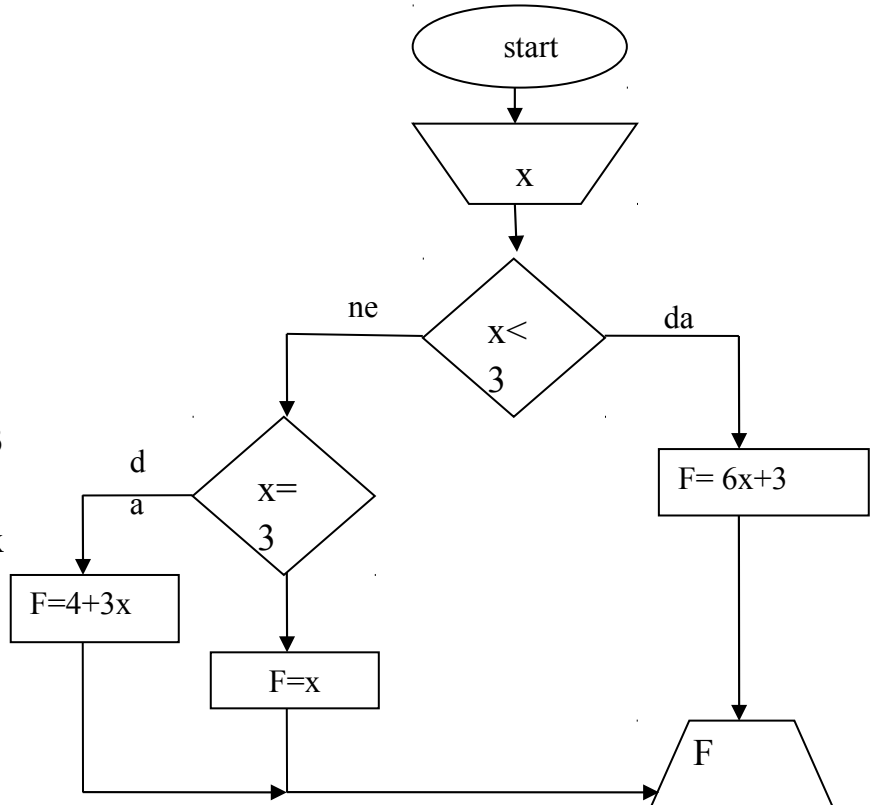
Решење:

```
Program z2;
Var obim,a,b,c,d,e: real;
Begin
Read (a,b,c,d,e);
obim:=a+b+c+d+e;
Write(obim);
End.
```



2. Написати алгоритам и програм за израчунавање:

$$F=\begin{cases} 4+3x; & x<3 \\ 6x+3; & x=3 \\ x, & x>3 \end{cases}$$

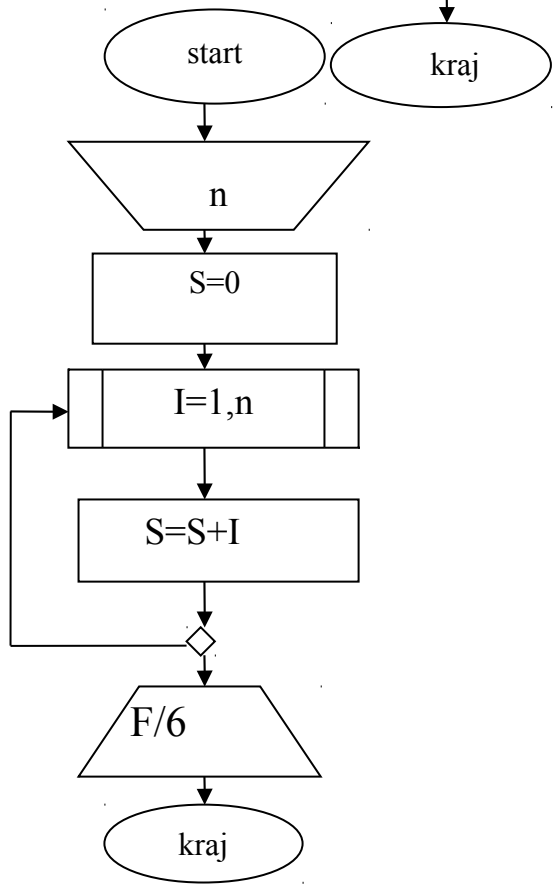


<u>Решење:</u>

```
Program z3;
Var F,x: real;
Begin
Read (x);
If x<3 then F:=6*x+3
Else
If x=3 then F:=4+3*x
else F:=x;
Write(F);
End.
```

3. Написати алгоритам и програм за израчунавање датог израза:

$$K=\frac{1+2+3+\ldots+n}{6}$$



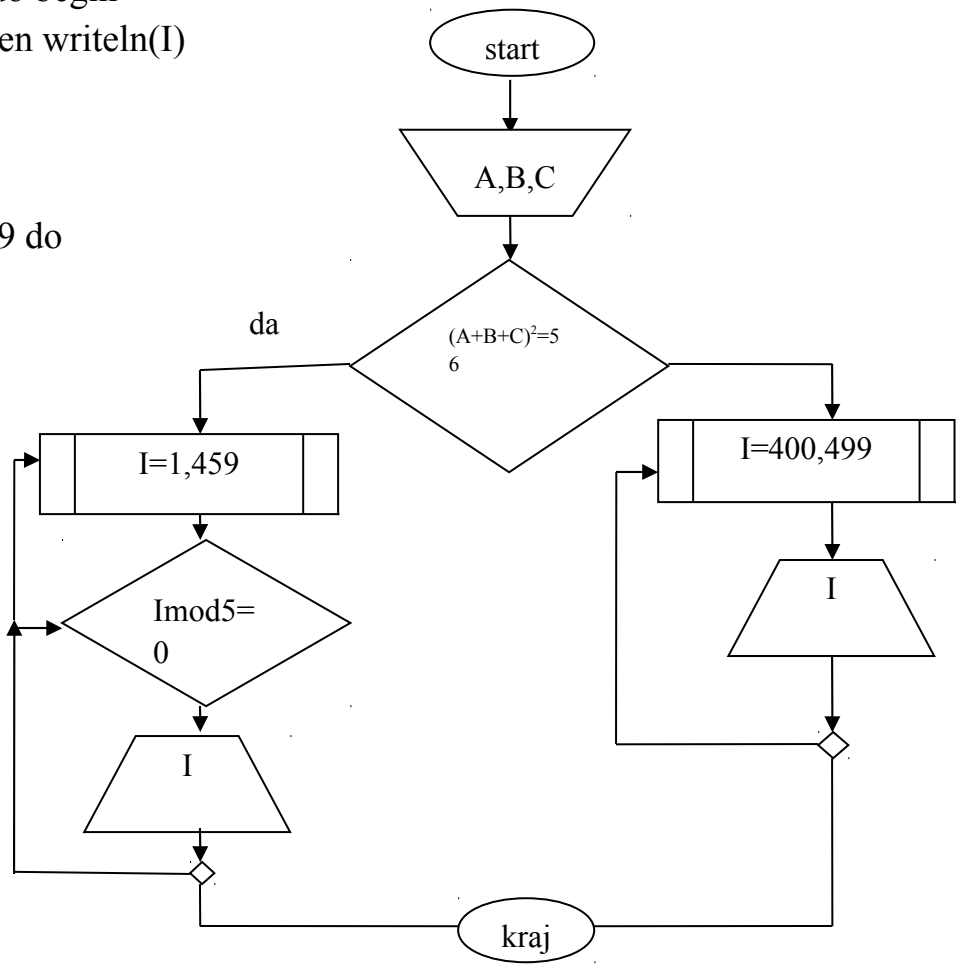
<u>Решење:</u>

```
Program z4;
Var I,n:integer, F:real;
Begin
Read (n);
S:=o;
For I:=1 to n do S:=S+I;
Write (F/6);
End.
```

4. Написати алгоритам и програм којим се уносе три броја, ако је њихов збир подигнут на квадрат једнак броју 56, показати све природне бројеве до 459 који су дељиви са 5, у супротном показати све троцифрене бројеве који почињу цифром 4.

Решење:

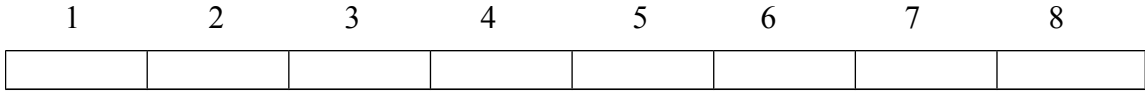
```
Program z5;
Var A,B,C: real;
I:integer;
Begin
Read (A,B,C);
If sqr(A+B+C)=56 then begin
For I:=1 to 459 do begin
If I mod 5 = 0 then writeln(I)
End;
End
Else begin
For I:=400 to 499 do
Write(I);
End;
End.
```



6. Низови

6.1. Једнодимензионални низови

Низ је скуп елемената који имају исто име, истог су типа, а разликују се само по индексу.



N Низ X дат у примеру има осам елемената.

У математици елеменат низа представља се: x_1, у Pascal-у x[1]. Дакле, индекс елемента низа потребно је уписати између угластих заграда.

Раније је било речи о основним типовима података, то су: integer, real, boolean и char.

Нови типови података дефинишу се наредбом TYPE, ако постоји у програму пише се после службене речи program, а пре var.

TYPE niz=ARRAY [1..максимална дужина низа] OF TIP PODATKA;

Пример: дат је целобројни низ од 100 елемената

TYPE niz=ARRAY [1..100] OF integer;

Пример: дат је низ реалних бројева од 234 елемента

TYPE niz=ARRAY [1..234] OF real;

Ако је непознат број елемената мора се унети конкретна вредност, како би Pascal знао колико простора да резервише у меморији.

Пример: дат је низ од n елемената

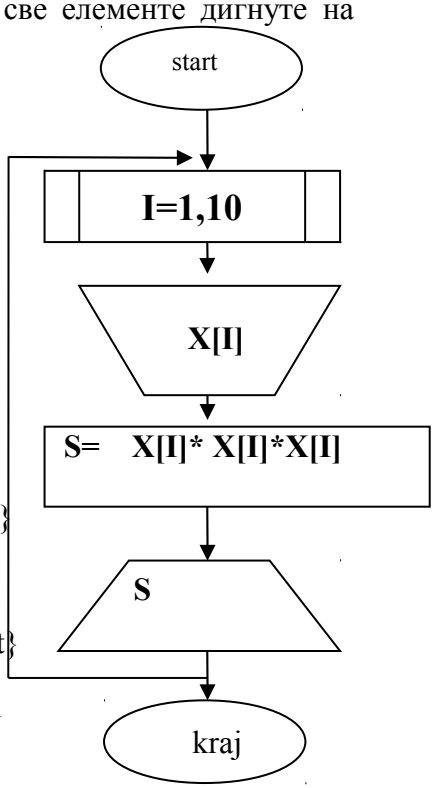
TYPE niz=ARRAY [1..10000] OF real;

Било какве радње које се врше над елементом низа подразумевају елеменат X[I], ако је потребно радити са индексом (редним бројем места елемента у низу) користи се само I.

Када се решава задатак, најчешће се најпре уносе сви елементи низа, а затим врши њихова обрада.

Пример:

Дат је целобројни низ од 10 елемента, показати све елементе дигнуте на трећи степен.



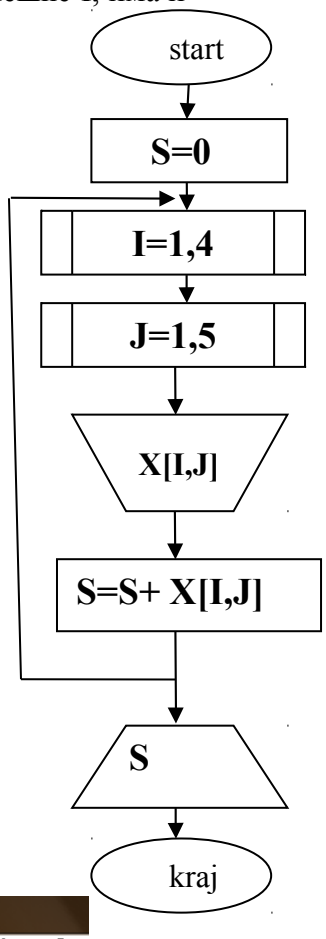

```
{ --- niz ZBIR ELEMENATA----- }
program zbirel;
Type niz=array[1..30]of integer;
var     I:integer;
        x:niz;
s:longint;
BEGIN
FOR I:=1 TO 10 DO
BEGIN
Write('Unesi X[',I,']='); {poruka se ispisuje na ekranu}
Readln(X[I]);  {unose se elementi niza}
S:= X[I]* X[I]* X[I]; {izracunava se trci stepen}
Write ('Treci stepen X[',I,'] je:',s); {ispisuje se rezultat}
                End;
WRITELN('Pritisni ENTER za nastavak...'); READLN
end.
```

```
Turbo Pascal 7.0
Turbo Pascal  Version 7.0  Copyright (c) 1983,92 Borland International
Unesi X[1]=2
 Treci stepen X[1] je:8
Unesi X[2]=12
 Treci stepen X[2] je:1728
Unesi X[3]=2
 Treci stepen X[3] je:8
Unesi X[4]=3
 Treci stepen X[4] je:27
Unesi X[5]=4
 Treci stepen X[5] je:64
Unesi X[6]=5
 Treci stepen X[6] je:125
Unesi X[7]=6
 Treci stepen X[7] je:216
Unesi X[8]=15
 Treci stepen X[8] je:3375
Unesi X[9]=7
 Treci stepen X[9] je:343
Unesi X[10]=11
 Treci stepen X[10] je:1331
Pritisni ENTER za nastavak...
```

6.2. Дводимензионални низови – Матрице

Матрица је правоугаона шема бројева.

Матрица је дводимензионални низ, који поред ознаке реда, најчешће I, има и ознаку колоне J. Елеменат матрице представља се X[I,J].



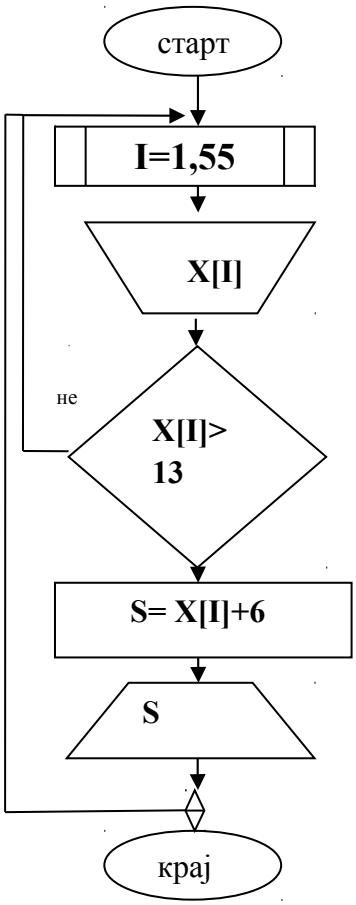
Пример: Израчунати збир свих елемената матрице 4x5

```
{ --- matrica ZBIR ELEMENATA----- }
Program zbirel;
Type mat=array[1..4,1..5]of integer;
Var J,I,S:integer; x:mat;
Begin
S:=0;
For I:=1 to 4 do begin
For J:=1 to 5 do begin
Write('Unesi [',I,',',j,']='); {poruka se ispisuje na ekranu}
Readln(X[I,J]);  {unose se elementi matrice}
S:=S+ X[I,J]       {sabiraju se elementi matrice}
End;end;
writeln('Zbir el. matrice je:',S);
WRITELN('Pritisni ENTER za nastavak...'); READLN
end.
```

```
Turbo Pascal 7.0
Turbo Pascal  Version 7.0  Copyright (c) 1983,92 Borland International
Unesi [1,1]=23
Unesi [1,2]=2
Unesi [1,3]=34
Unesi [1,4]=123
Unesi [1,5]=23
Unesi [2,1]=23
Unesi [2,2]=2
Unesi [2,3]=90
Unesi [2,4]=234
Unesi [2,5]=1
Unesi [3,1]=2
Unesi [3,2]=4
Unesi [3,3]=5
Unesi [3,4]=6
Unesi [3,5]=7
Unesi [4,1]=8
Unesi [4,2]=9
Unesi [4,3]=123
Unesi [4,4]=234
Unesi [4,5]=3
Zbir el. Matrice je:956
Pritisni ENTER za nastavak...
```

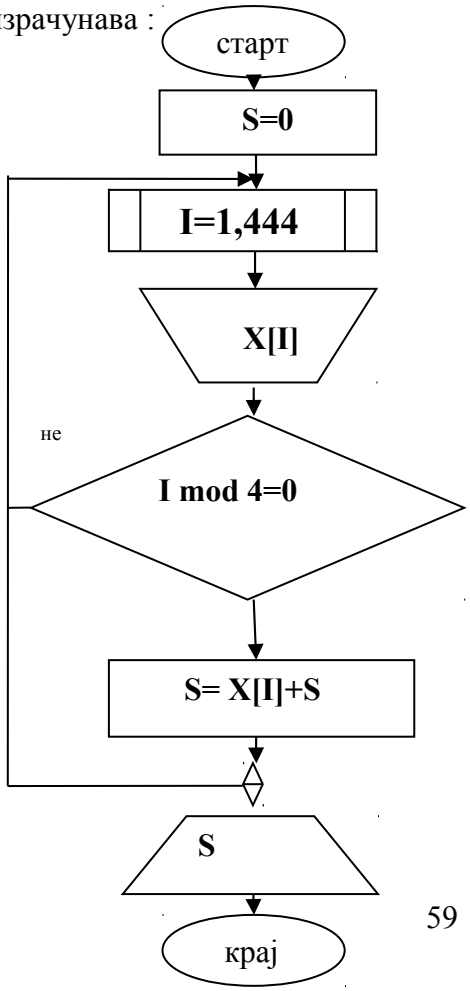
Задаци – пример контролног задатка

1. Написати алгоритам и програм којим се сви елементи низа већи од 13 увећавају за 6 (низ има 55 елемената).



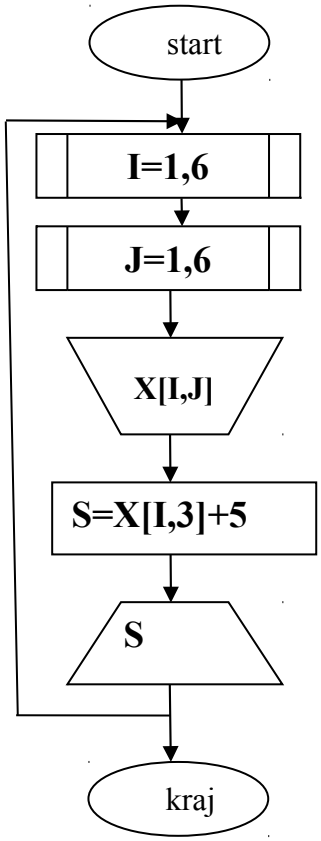

```
Program z1;
Type niz=array[1..55]of real;
Var x:niz;
I:integer; S:real;
Begin
For I:=1 to 55 do begin
Read ( X[I]);
If  X[I]>13 then begin
S:= X[I]+6;
Writeln('S=', S);
End;
End.
```

2. Написати алгоритам и програм којим се израчунава :
S = x[4]+x[8]+x[12]+ ... +x[444]



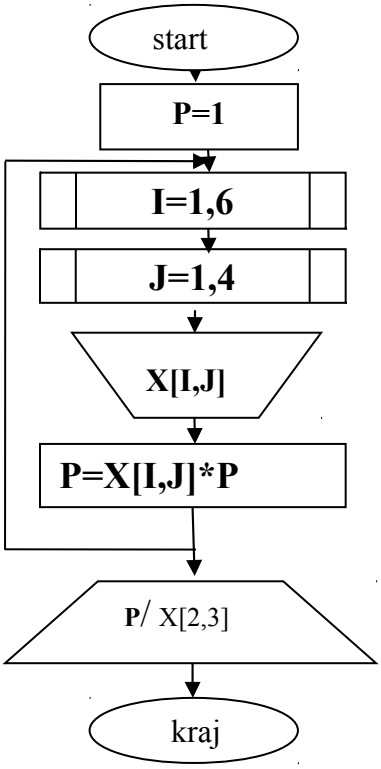

```
Program z2;
Type niz=array[1..444]of real;
Var x:niz;
I:integer; S:real;
Begin
For I:=1 to 444 do begin
Read ( X[I]);
If  I mod 4 =0 then
S:= X[I]+S;
End;
Writeln('S=', S);
End.
```

3. Написати алгоритам и програм којим се сви елементи матрице 6x6 који се налазе у трећој колони увећавају за 5, а затим штампају.




```
Program z3;
Type matrica=array[1..6,1..6]of real;
Var x:matrica;
I,J:integer; S:real;
Begin
For I:=1 to 6 do begin
For J:=1 to 6 do begin
Read ( X[I,J]);
S:= X[I,3]+5;
Writeln('S=', S);
End;
End.
```

4. Написати алгоритам и програм којим се производ свих елемената матрице 6x4, дели елементом који се налази на пресеку треће колоне и друге врсте.




```
Program z4;
Type matrica=array[1..6,1..4]of real;
Var x:matrica;
I,J:integer; P:real;
Begin
P:=1;
For I:=1 to 6 do begin
For J:=1 to 4 do begin
Read ( X[I,J]);
P:= P*X[I,J];
End;
Writeln('P=',P/X[2,3]);
End.
```

7. Подпрограми

Уколико се у програму одређени број наредби понавља, или представља засебну целину - приступа се писању подпрограма. Pascal има две врсте подпрограма: функције и процедуре.

Функције се користе у ситуацијама када је потребно израчунати једну вредност, док се процедуре користе у свим другим случајевима, две или више вредности, размене вредности и слично. Подпрограм, ако постоји у програму умеће се после наредбе за декларисање променљивих VAR, а пре BEGIN. Подпрограм се пише једном, а може се позивати више пута, зависно од потребе.

7.1. Функцијски подпрограм

Општи облик функције гласи:

Function ИмеФункције (ФиктивнеУлазнеВеличине:Тип Податка):Тип Резултата;

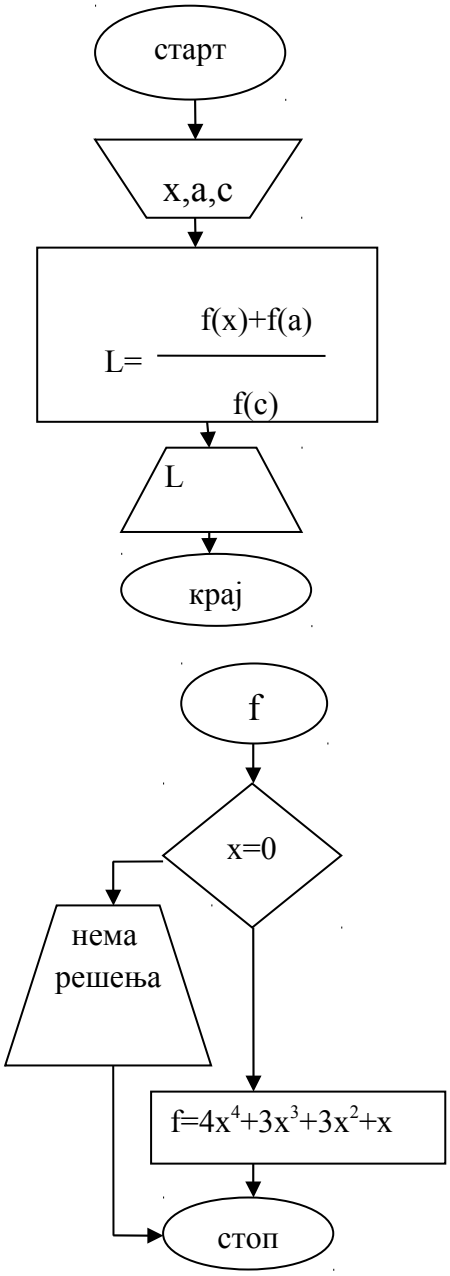
За дефинисање имена функције важе иста правила као за дефинисање имена програма (не може почињати бројем, имати специјалне знаке, бити службена реч Pascal-a и слично). Приликом извршавања главног програма на место намењено фиктивним улазним величинама доћи ће конкретна вредност, зависно од задатка. Тип податка може бити било који основни тип програмског језика Pascal, као и неки изведени, исто важи и за тип резултата.

Пример:

Саставити програм за израчунавање датог израза користећи функцијски подпрограм.

$$L=\frac{4x^4+3x^3+3x^2+x \;+\; 4a^4+3a^3+3a^2+a}{4c^4+3c^3+3c^2+c}$$

```
program z1;
var x,a,c, L:real;
 function f(x:real):real;
 begin
 if x=0 then write
 (’nema resenja’)
  else begin
   f:=4*sqr(x)*sqr(x)+3*x*sqr(x)+3*sqr(x)+x;
   end; end;
begin
writeln (’Unesi vrednosti za x,c i a’);
readln (x,c,a);
L:=(f(x)+f(a))/f(c);
writeln (’L=’,L);
end.
```



Најпре је потребно уочити да ли постоје елементи који се понављају, у примеру је то конструкција: четири пута нешто на четврти степен, плус три пута нешто на трећи степен, плус три пута нешто на квадрат, плус нешто. У програму ћемо ову конструкцију написати једном, а позвати је три пута у главном програму за различите улазне вредности.

7.2. Процедуре

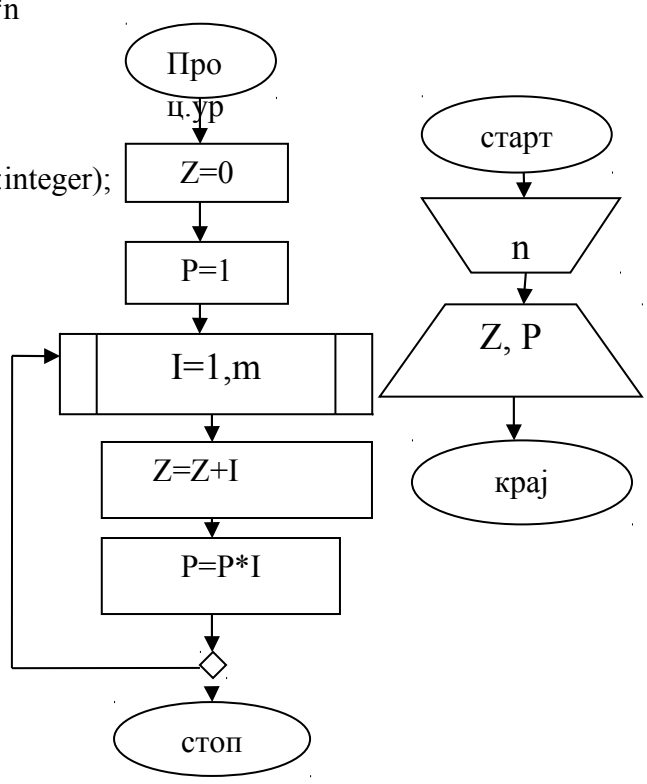
Рад са процедурама сличан је раду са функцијама. Већ је наглашено подпрограму процедуралног типа приступа се када је потребно радити са више излазних величина. Баш из тог разлога тип податка излазне величине синтаксно се не записује на крају основне наредбе Function, већ се дефинише у оквиру дела за параметре:

Procedure Имепроцедуре (ФиктивнеУлазнеВеличине: тип податка; var Фиктивне Излазне Величине: тип резултата);

За дефинисање имена процедуре важе иста правила као за дефинисање имена програма (не може почињати бројем, имати специјалне знаке, бити службена реч Pascal-a и слично). Приликом извршавања главног програма на место намењено фиктивним улазним величинама доћи ће конкретна вредност, зависно од задатка. Тип податка може бити било који основни тип програмског језика Pascal, као и неки изведени, исто важи и за тип резултата. Разлика између процедура и функција је и у позиву. Функција се позива у оквиру израза, док се процедура може позвати самостално.

Пример: Процедуром израчунати збир и производ:
Z=1+2+...+n P=1*2*...*n

```
Program z1;
var n,I,Z,P:integer;
  Procedure proc(m:integer; var Z1,P1:integer);
  var I:integer;
  begin
  Z1:=0;P1:=1;
  for I:=1 to m do begin
  Z1:=Z1+I;
  P1:=P1* I;
  end;
  end;
begin
write('Unesi broj');
readln(n);
writeln(proc(n,Z,P));end.
```



8. Слогови

Термин слог налази примену у свакодневном животу. Додуше може имати више значења. Тако, у програмирању се под слогом подразумева група података који имају неку заједничку особину, а разликују се најчешће према типу податка.

Слог може бити ученик, његови подаци у програмирању представљају поља или атрибуте. Једно поље је име и презиме, пошто је реч о текстуалном податку његов тип ће бити String, такође и адреса припада овом типу податка. Али зато, просек оцена припада типу Real, број предмета које похађа био би тип Integer и слично. Употреба слогова умногоме олакшава рад када је потребно радити са пакетима података.

Општи облик дефиниције слога гласи:

```
Type  ImeSloga=Record
      Polje 1: TipPodatka;
      Polje 2: TipPodatka;
      Polje 3: TipPodatka;
        .
        .
        .
      Polje n: TipPodatka
                  End;
```

Пример:

```
Program Z1;
Type Ucenik=Record
      Ime:string;
      Prezime:string;
      Adresa:string;
      Razred:integer;
      Odeljenje:integer;
      Prosek:real
            End;
Var  Ucenik1,Ucenik2:Ucenik;
```

У примеру је дефинисан слог ученик са 6 поља, атрибута. Сваком пољу појединачно може се приступити навођењем имена променљиве која је типа слог и имена поља који се раздвајају тачком.

Ucenik1.Ime – приступамо податку који је типа слог – подтипа стринг.

Слогу се приступа само помоћу наредбе доделе: Ucenik1.Ime:='Petar';

Резултат функције не може бити слог.

Пример: Саставити програм којим се дефинише слог град, са најмање четири поља и унети податке за један град:

```
Program z1;
Type grad=record
        Naziv:string;
        Broj_naselja:integer;
        Povrsina:real;
        Okrug:string
        End;
Var grad1:grad;
Begin
Grad1.Naziv:=’Zrenjanin’;
Grad1.Broj_naselja:=22;
Grad1.Povrsina:=1,324;
Grad1.Okrug:=’Srednje-banatski’;
End.
```

Слогове можемо користити као компоненте других структурних типова, као што су то низови:

```
Type
 gradjani=array[1..132051] of grad;
```

наравно под условом да ово поље постоји дефинисано у слогу.

```
Var      stanovnik:gradjani;
```

8.1. Наредба WITH... DO

Наредбом WITH омогућује се поједностављење записа позива слога.

У претходном примеру неколико пута је поновљен запис „Grad1“:

```
Grad1.Naziv:=’Zrenjanin’;
Grad1.Broj_naselja:=22;
Grad1.Povrsina:=1,324;
Grad1.Okrug:=’Srednje-banatski’;
```

Помоћу наредбе WITH запис ће изгледати другачије:

```
With Grad1 do begin
Naziv:=’Zrenjanin’;
Broj_naselja:=22;
Povrsina:=1,324;
End;
```

9. Датотеке

У свим програмима до сада резултати рада нестајали су након извршења програма, за одређене улазне величине добијао се одређени резултат. Корисник је видео добијени резултат, али није био у стању да га користи и у решавању неког другог задатка. Да би се обезбедило чување резултата и његова поновна употреба, потребно је сачувати га на неком спољашњем носиоцу информација. Постоји више начина овакве организације података, једноставнији али и са већим ограничењима је употреба датотека, сложенији али доступан различитим програмима је организација у базе података. Област база података веома је развијена и представља целину у свету рачунарства, најпознатије базе су : Microsoft Access, Oracle, My SQL итд.

Појам датотеке у Паскалу представља део меморије (са својим именом) на неком спољном носиоцу информација, који може чувати одређен број података. Број зависи од могућности конкретног меморијског медијума. Операција уписа у датотеку означава пренос података из оперативне меморије у датотеку, а операција читања означава пренос садржаја из датотеке у оперативну меморију.

```
TYPE NazivDatoteke=file of TipPodataka;
VAR a: NazivDatoteke;
```

Или

```
VAR
a: file of TipPodataka;
```

a је датотечна променљива, не може се користити у наредби доделе нити у другим наредбама где се обично користе променљиве. Она се посматра као представник датотеке над којом се врши рад.

9.1. Основне операције са датотекама

Већ је наглашено да се унапред не зна величина датотеке, тако да је манипулација подацима најједноставнија употребом наредбе за крај. Крај се обележава маркером – **EOF (end of file).** Ова наредба користи се у склопу наредбе While... do.

Први корак у раду је повезивање датотечне променљиве са конкретном физичком датотеком на диску, то се постиже наредбом:

ASSIGN(DatotecnaPromenljiva,ImeFajla), уместо имена фајла може се навести и пут до датотеке.

Креирање нове датотеке, односно њено отварање за упис постиже се процедуром:

REWRITE(ImeDatotecnePromenljive)

Овом процедуром настаје увек празна датотека, ако је постојала раније датотека са тим именом њен садржај ће бити обрисан.

Упис у датотеку врши се наредбом:

WRITE(Datotecna Promenljiva,Promenljiva)

Promenljiva је истог типа као променљиве које чине датотеку, уместо једне може се навести и више променљивих.

Овом процедуром подаци се не шаљу директно на диск, прво иду у бафер (специјални део оперативне меморије), и тек кад се бафер напуни онда се врши преношење података.

Датотека се отвара за читање процедуром:

RESET(ImeDatotecnePromenljive)

Датотечни показивач поставља се на нулу.

Читање података реализује процедура:

READ(DatotecnaPromenljiva, Promenljiva);

Ова процедура се може више пута применити над једном датотеком, ако је она била отворена за читање, аутоматски ће се затворити.

На крају рада обавезно се користи процедура:

CLOSE(ImeDatotecnePromenljive);

Постоји више врста датотека у Паскалу:

- Текстуалне (TEXT)
- Типизиране (file of Tip)
- Нетипизиране (описују се типом file)

Задаци – рад са датотекама

1. Написати програм којим се у нову датотеку бројева ’нова.дат' уписују бројеви између 23 и 123 дигнути на трећи степен.

У већини задатака долазак до решења знатно је олакшан креирањем алгоритама. Међутим у раду са слоговима и датотекама алгоритам нема неки већи значај, те се веома ретко користи.

```
Program z1;
Var     f:file of integer;
        I:integer;
Begin
Assign (f,’nova.dat’);
Rewrite(f);
For I:=23 to 123 do begin writeln(f,I*I*I); end;
Close (f);
End.
```

2. Написати програм којим се у нову датотеку ’нова.дат' уписују сва имена из постојеће датотеке ’стара.дат’, која почињу словом ’К’.

```
Program z2;
Var     f,g:file of string;
        s:string;
Begin
Assign(f, ’nova.dat’);
Assign(g, ’stara.dat’);
Reset(g);
Rewrite(f);
While not eof(g) do begin
Read(g,s);
If s[1]=’K’ then writeln(f,s);
End;
Close(f);
Close(g);
End.
```

3. Написати програм којим се у нову датотеку ’нова.дат' уписују сви бројеви из старе датотеке ’стара.дат’, који су парни и дељиви са 3.

```
Program z3;
Var     f,g:file of integer;
        n:integer;
Begin
Assign(f, ’nova.dat’);
Assign(g, ’stara.dat’);
Reset(g);
Rewrite(f);
While not eof(g) do begin
Read(g,n);
If (n mod 2=0) and (n mod 3=0) then writeln(f,n);
End;
Close(f);
Close(g);
End.
```

4. Написати програм којим се у нову датотеку ’нова.дат' уписују сва имена из постојеће датотеке ’стара.дат’, која имају дужину 6 знакова.

```
Program z4;
Var     f,g:file of string;
        s:string;
Begin
Assign(f, ’nova.dat’);
Assign(g, ’stara.dat’);
Reset(g);
Rewrite(f);
While not eof(g) do begin
Read(g,s);
If length(s)=6 then writeln(f,s);
End;
Close(f);
Close(g);
End.
```

Литература

1. Милан Чабаркапа „Рачунарство и информатика за 3. Разред гимназије“, Круг, 2004. Београд
2. Проф. др Душан Малбашки „Одабрана поглавља метода програмирања“, УНС 2002. Нови Сад
3. Замуровић мр Снежана, Замуровић Љубомир „Информатички азбучник“, ауторски репринт, 2008. Кикинда

Електронска литература

http://portal.acm.org/

http://www.swissdelphicenter.ch/en/niklauswirth.php

http://www.marcocantu.com/epascal/English/ch01hist.htm

http://www.tiobe.com/index.php/content/paperinfo/tpci/

http://www.moorecad.com/standardpascal/iso7185.pdf

http://www.lysator.liu.se/c/bwk-on-pascal.html

http://soft.softodrom.ru/ap/Pascal-ABC-p6425