Pascal / Delphi

Pascal је развијен од стране швајцарског професора Niklaus Wirth-а. Wirth је дао име свом језику по француском физичару, математичару и филозофу Blaise Pascal-у, творцу прве механичке рачунске машине за сабирање. Pascal је представљен 1965. године, прва верзија језика је објављена 1968. године, са преводиоцем (compiler) написаним у FORTAN-у (Formula Translator). Друга верзија преводиоца написана је у самом Pascaly, и објављена је 1971. године. Напокон, 1974. године објављена је сређена верзија која се узима као стандард, и прави почетак Pascal језика.

САДРЖАЈ:

I. Увод у програмирање ... 3
1. Програм, програмирање и програмски језици ... 3
2. Процес програмирања ... 4
3. Алгоритми ... 5
II. Структура и синтакса Delphi / Pascal програма: ... 6
1. Структура главног пројектног модула (*.dpr фајл) ... 6
2. Структура и синтакса Unit-а (*.pas фајл) ... 7
III. Основни елементи програмског језика Pascal ... 8
1. Основни симболи Pascal језика ... 8
2. Основни синтаксни елементи ... 8
А. Коментар ... 8
Б. Идентификатори ... 8
В. Сепаратори ... 9
Г. Стандардна имена ... 9
Д. Бројеви ... 9
Ђ. Стрингови ... 9
IV. Изрази и оператори ... 9
1. Аритметички оператори ... 9
2. Релациони оператори ... 9
3. Логички boolean оператори ... 9
4. Редослед важности оператора ... 9
V. Декларација константи и променљивих ... 10
1. Константе ... 10
2. Променљиве ... 10
3. Локалне и глобалне променљиве ... 10
VI. Типови података ... 11
1. Целобројни типови ... 11
2. Реални типови података ... 11
3. Логички тип (Boolean) ... 11
4. Знаковни тип (Char) ... 12
5. Стринг тип (String) ... 12
VII. Контрола тока програма, подела наредба ... 13
1. Просте наредбе ... 13
А. Наредба доделе ... 13
Б. Позив процедуре ... 13
В. Goto наредба ... 13
Г. Празна наредба ... 13
2. Сложене – структуиране наредбе ... 13
А. Саставна – блок наредба ... 13
Б. Условне наредбе – IF ... 14
В. Понављајуће наредбе – структуре циклуса ... 17
VIII. Процедуре и функције ... 18
1. Декларација функција ... 18
2. Декларација процедуре ... 18

I. Увод у програмирање

1. Програм, програмирање и програмски језици

Програм је скуп инструкција којима говоримо компјутеру шта тачно да ради. Апликација је још једно име за програм. Програмирање представља процес креирања или писања инструкција.

Ако хоћемо да компјутер уради тачно оно што желимо, морамо да набавимо програм. Компјутер (хардвер) не ради ништа сам. У суштини компјутер је обичан електронски уређај без интелигенције. Компјутер не размишља. Само слепо прати људске инструкције у виду програма.

Претпоставимо да хоћемо да отворимо видео клуб. Желимо да нам компјутер евидентира чланове клуба, раднике, прати процес рентирања, трошкове, зараду и сл. Где наћи програм?

Постоје три начина:

1. Купити неки постојећи и надати се да програм ради тачно оно што нам треба;
2. Написати свој сопствени програм;
3. Купити програм са изворним кодом (source code).

Код је друго име за програм.

1. Најлакше, најбрже а обично и најјефтиније је да се купи готов програм. Постоји десетине хиљада програма на тржишту, за готово све области људског деловања и интересовања. Проблем овог приступа је што обично купљени програм не задовољава 100% захтевима корисника.
2. Ипак људи купују компјутере да би радили баш оно што им треба. Фирме не могу да мењају тотално свој начин рада. Зато обично фирме направе свој сопствени програм, који се потпуно уклапа у њихов дотадашњи начин рада. Претходно, дакле не важи само за предузећа већ и за научнике, уметнике, људе свих осталих делатности. Нема двоје људи који води своје кућне финансије на исти начин. Сви они траже специјализован програм за њихове потребе. Мане овог приступа су високи трошкови.
3. Неки програми су изменљиви, прилагодљиви. Трећи метод је куповина програма са ***изворнм кодом***. Преко изворног кода може се извршити модификација програма у жељеном правцу. Нажалост већина комерцијалног софтвера не даје и изворни код. Већина програма се преводи, после чега је изворни код закључан у извршни код који се не може мењати.

Уколико се определимо за писање програма, потребан нам је одговарајући алат. Тачније, потребан је програмски језик.

Језик је средство за представљање и преношење информација, као и за комуникацију између два или више корисника.

Корисник тако може бити човек или рецимо компјутер.

Језике можемо поделити на природне и вештачке. Природне језике користе људи за успостављање међусобне комуникације. Постоји више стотина језика ове врсте, као што су: српски, енглески, француски, шпански, немачки, кинески и сл. Вештачки језици су створени за потребе приказивања информација као што су записивање формула из математике, физике, хемије и сл.

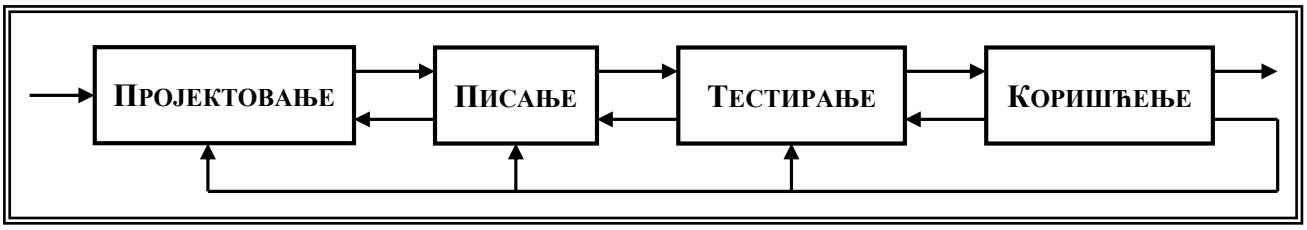
Што се тиче комуникације између човека и компјутера, проблем је што компјутер разуме само свој машински језик, у виду бесконачне секвенце нула и јединица – 11000111 11100110 10000001.

На жалост (или на срећу) овакав начин комуникације је за човека исувише тежак и мукотрпан. За комуникацију између човека и компјутера и записивање алгоритма створени су програмски језици. Програмски језици су углавном језици састављени од симбола енглеског алфабета, педесетак кључних речи (такође узетих из енглеског) и још неких специјалних симбола. Овако формиран језик изгледа знатно прихватљивије за човека, али је потпуно неприхватљив за компјутер. Зато уз програмске језике иду преводиоци, посебни програми који претварају текст који личи на енглески у серију нула и јединица, које су разумљиве компјутеру. Постоји више десетина (стотина) програмских језика од којих су неки од најпознатијих: C++, Java, Basic, Pascal, Cobol, Fotran итд.

2. Процес програмирања

Генерално процес програмирања је један сложен процес који се састоји од четири главне фазе:

1. Пројектовање програма
2. Писање
3. Тестирање
4. Коришћење програма



Дијаграм 1. Процес програмирања

I. **Пројектовање програма.** У овој фази се врши идентификација проблема који треба да се реши, као и његова детаљна анализа.Треба решити које ће све функције програм да обавља, затим питање улазно-излазних података који ће се користити у програму, питање складиштења или чувања тих података, модуларност и структура програма, дизајн корисничког интерфејса и сл. Уколико се користи процедурално програмирање, обично се користи методологија ''одозго на доле'', где се читав програм дели на модуле, тако што су на врху хијерархије модули везани за крупније заокружене логичке целине, које се даље деле на додатне модуле који су ниже у хијерархији итд. Уколико се користи нпр. објектно орјентисана методологија, онда се врши идентификација објеката (нпр. у некој игри имали би смо можда објекте диносаурусе, тираносаурусе и сл.), њихов међусобни однос, особине итд. За решавање конкретних проблема најчешће се користе алгоритми. У овој фази се обично врши и избор програмског језика који ће се користити.

II. **Писање програма.** Ова фаза се састоји од следећих процеса:

1. **Едитовање** (Editing) – писање изворног кода;
2. **Превођење** (Compiling) – изворни код се пребацује у бинарни облик и проверава синтаксна исправност програма;
3. **Повезивање** (Linking) – сви модули који чине програм повезују се у једнинствену целину.
4. **Прављење извршног фајла** (Making.exe file) – Прави се самостални извршни фајл који може да се покрене самостално унутар оперативног система за који је предвиђен;

Током читаве ове фазе разликујемо процес **откривања грешака** или дебаговања (debugging). Синтакса се односи на правопис и граматику језика. Баг (bug) је програмска грешка.

Разликујемо три врсте грешака: (1) **Синтаксне**; (2) **Логичке** грешке; (3) Грешке **намере**.

Следећа фраза демонстрира прве две: “Постоје две гррешке у овој реченици.”

Прва грешка је синтаксна, јер је грешка погрешно написана. Друга грешка је логичка и захтева више времена за проналажење. Логичка грешка је да је реченица логички нетачна: постоји само једна грешка – синтаксна, а не две.

Примери грешака намере су покушај учитавања података са драјва који није спреман (дискета, диск нису убачени у драјв), или покушај приказивања слике која је избрисана или премештена и сл.).

III. **Тестирање програма.** У овој фази треба испитатати програм у условима његовог рада, тако што се симулирају све могуће комбинације улазних података, и контролише вредност излазних резултата који треба да одговарају реалним вредностима.

IV. **Коришћемје програма.** Из ове фазе имамо два излаза: један је крај програма, а други улаз у нови циклус развоја програма.

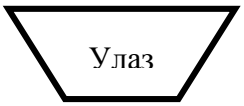
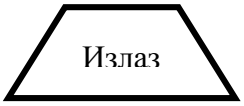
3. Алгоритми

Алгоритам представља скуп упутстава за извршење неке активности или операције.
Алгоритам за неку класу задатака представља коначан скуп правила помоћу којих решење сваког проблема из те класе може бити нађено аутоматским путем.

Решење проблема, рецептура. Програм је алгоритам који компјутер може да обради.

Појединачно упутство назива се корак алгоритма. Сваки корак мора да буде еквивалент једној познатој операцији.

Обично се за представљање алгоритма користе блок дијаграми (алгоритамска шема, дијаграм тока и сл.), где се сваки корак представља графичким симболом.

Графички симбол корака	**Функција корака**
	Почетак алгоритма.
	Улазне величине алгоритма
	Обрада података
	Условни корак – гранање
	Крај циклуса
	Излазне величине алгоритма
	Крај алгоритма

II. Структура и синтакса Delphi / Pascal програма:

Програм је састављен од модула са изворним кодом званих units. Сваки unit има свој фајл и преводи се засебно. Преведени unit-и се повезују да би креирали програм/апликацију.
Унити нам дозвољавају да:

- Поделимо велике програме у модуле који могу да се едитују посебно.
- Креирамо библиотеке (libraries) које можемо делити између више програма.
- Дистрибуирамо библиотеке другим програмерима без откривања изворног кода.

Borland алати користе пројекатни (.dpr) фајл за чување „главног" програма, док углавном већина изворног кода се чува у unit (.pas) фајловима. Свака апликација се састоји од једног пројектног фајла и једног или више unit фајлова.

1. Структура главног пројектног модула (*.dpr фајл)

Програм се састоји од:

- Заглавља програма (прецизира име програма)
- Клаузуле uses (опционо) која наводи листу коришћених unit-а , и
- Блока декларација и изјава (наредби) које се извршавају покретањем програма.

<u>Типичан пројектни фајл/модул:</u>

```
1. program Elementi;                                  // Zaglavlje programa
2. uses                                               // Klauzula uses
3.    Forms, UElementi in 'UElementi.pas' {FGlavna};
4. {$R *.res}
5. Begin                                              //Blok naredbi
6.    Application.Initialize;
7.    Application.CreateForm(TFElementi, FElementi);
8.    Application.Run;
9. end.
```

Заглавље програма прецизира име програма, састоји се од кључне речи program праћене исправним идентификатором и тачка зарезом. Идентификатор мора да се поклопи са именом проектног фајла (у нашем случају то би био Elementi.dpr).
Uses клаузула наводи листу свих unit-а кои су инкорпорирани у сам програм. Ови unit-и могу такође да имау своју uses клаузулу.
Блок се састоји од простих или структуираних изјава које се извршавају покретањем програма. Уоквирене су кључним речима begin и end. Линије 6 до 9 позивају методе пројектног Application објекта (иницијализација, креирање прозора и приказивање.
Такође овде могу да се нађу декларације константи, типова, променљивих, процедура и функција. Ове декларације морају да претходе делу са наредбама.

2. Структура и синтакса Unit-а (*.pas фајл)

Unit се састоји од типова (укључивши класе), константи, променљивих и рутина (процедура и функција). Сваки unit је дефинисан унутар свог (.pas) фајла. Почиње са unit заглављем, праћеног interface, implementation, initialization, и finalization секцијама. Initialization и finalization секције су опционе.

```
1.  unit Unit1;
2.  interface
3.  uses  { List of units goes here }
4.  { Interface section goes here }
5.  implementation
6.  uses  { List of units goes here }
7.  { Implementation section goes here }
8.  initialization
9.  { Initialization section goes here }
10. finalization
11. { Finalization section goes here }
12. end.
```

Унит се мора завршити са кључном речи end праћеном тачком.
Заглавље унита прецизира име унита, и састои се од резервисане речи unit, праћене валидним идентификатором и тачка зарезом. Идентификатор мора да одговара имену unit фајлу. Имена унита морају да буду јединствена унутар програма.

Интерфејс секција почиње кључном речи interface и наставља се све до почетка секције имплементације (implementation). Овде се декларишу константе, типови, променљиве, процедуре и функције које су доступне клијентима (другим унитима и програмима). Ови елементи се називају јавним (public). Декларација рутина садржи само њихово заглавље, док блок наредби рутине следи у секцији имплементације (implementation). Такође може да има своју сопствену uses клаузулу, која тада мора бити прва после речи interface.
Секција за имплементацију почиње са резервисаном речи implementation и наставља се до почетка секције за иницијализацију или до краја унита уколико нема секције за иницијализацију.

Насупрот јавним овде се могу декларисати константе, типови, променљиве, процедуре и функције које су приватне, илити само за интерну употребу.

Секција за иницијализацију је опциона, почиње са кључном речи initialization и наставља се до почетка секције за финализацију (finalization) уколико ова постоји или до краја унита у супротном.
Чине је наредбе које се извршавају на почетку програма, па је стога згодна рецимо за почетну иницијализацију одређених структура података.
Секција за финализацију је такође опциона и може се појавити само у унитима где постоји секција иницијализације. Почиње са резервисаном речи finalization и наставља се до краја унита.
Наредбе у овој секцији се извршавају када се главни програм затвара, и користи се да се ослободе ресурси који су алоцирани у секцији иницијализације.

III. Основни елементи програмског језика Pascal

1. Основни симболи Pascal језика

Основни симболи су недељиве јединице језика из којих се образују остале сложеније јединице. Скуп симбола Pascal језика састављен је од:

1. Великих и малих слова енглеске азбуке A..Z a..z
2. Цифара декадног бројног система 0..9
3. Специјалних симбола: # $ & ' ( ) * + , – . / : ; < = > @ [ ] ^ { } и парови симбола: **(* (. *) .) ..** // **:=** <= >= <>
4. Службених (кључних, резервисаних) речи: **Program, Uses, In, Begin, End, Unit, Interface, Implementation, Initialization, Finalization, Procedure, Var, Const, And, Case, ,Div, Do, Else, File, For, Function, If, Mod, Not, Object, Or, Record, Repeat, String, Then, To, Until, While** итд.

Кључне речи су сложенији али и даље елементарни симболи програмског језика.
Pascal импементиран у Delphi 7 верзији има укупно 73 кључних речи:

2. Основни синтаксни елементи

Правила која одређују како се над скупом симбола граде елементарне и сложене конструкције чине **граматику** језика. **Граматика** је наука о језику и његовим законима.

Синтакса је наука о језику која изучава образовање граматички коректних конструкција језика. Синтаксу језика чини скуп правила за образовање правилних конструкција језика. Синтаксне грешке се иначе откривају од стране самог преводиоца.

Семантика је наука о језику која изучава значење конструкција језика. Семантику језика чини скуп правила за утврђивање значења конструкција језика.

Уз помоћ основних синтаксних елемената (још их називамо и **токени**) формирамо сложеније форме као што су **изрази** (expressions), **декларације** (declarations) и **наредбе** / изјаве (statements). **Наредба** описује алгоритамску акцију која се извршава унутар програма. **Израз** је синтаксна јединица која се појављује унутар наредби и представља неку вредност. **Декларације** дефинишу идентификаторе који се могу користити унутар израза и наредби, и тамо где је потребно врше алокацију меморије.

НАРЕДБА ПРЕДСТАВЉА СИНТАКСНО КОМПЛЕТНУ ЈЕДИНИЦУ ПРОГРАМА КОЈА ИЗРАЖАВА ЈЕДНУ ВРСТУ АКЦИЈЕ, ДЕКЛАРАЦИЈЕ ИЛИ ДЕФИНИЦИЈЕ.

А. Коментар

Постоје три форме писања коментара:

- {Ovo je komentar}
- (* Ovo je jos jedan komentar*)
- // Ovo je komentar do kraja linije

Коментари се не узимају од стране преводиоца, и немају никаквог утицаја на ток извршавања програма.

Б. Идентификатори

У Паскалу све мора да има име, почевши од програма. То важи пре свега за константе, променљиве, типове, функције, процедуре. Правила за конструисање идентификатора:

- Састоји се од комбинације слова, бројева и знака за подвучено
- Први знак не може бити број
- Размак није дозвољен
- Не смеју се користити кључне или резервисане речи
- Употреба малих и великих слова не прави разлику, па су тако BtnHello, btnhello, btnHello и BTNHello идентични идентификатори.

В. Сепаратори

Сепаратори у програму одвајају имена, константе или кључне речи. Као сепаратор можемо користити коментар, размак (<SPACE>) или знак за крај реда (<RETURN>). Допуштен је произвољан број сепаратора.

Г. Стандардна имена

У Pascaly су дефинисана стандардна имена чије је значење унапред дефинисано, али се могу предефинисати у програму (није препоручљиво). Ту спадају уграђени типови података, процедура и функција. Нека од стандардних имена: Boolean, Char, False, Real, True, Abs, Round итд.

Д. Бројеви

Бројеви могу бити *цели* и *реални* (у покретном зарезу – floating point). Број може имати предзнак + или -, док се + не мора писати. **Цео број** записује се као низ цифара и не сме имати тачку.
Пример: 0, 1234, 07, -16, +987

Реални број садржи децималну тачку. Испред и иза тачке мора да буде бар једна цифра. Реални број се може писати и са децималним експонентом: 3.3E4. Испред слова E мора бити реални или цео број, а иза неки цео број (експонент). 3.3E4 = 3.3 * 10^4 = 33000.0
Пример: 0.0, +1234.0, -5.64, 3.1459, 26E-15, -2E6, 7.7E+15

Ђ. Стрингови

Стринг је низ знакова уоквирен једноструким наводницима. Једноструки наводник унутар стринга пише се као два наводника. *Пример:*‘‘, ‘Pascal’, ‘123456789’, ‘Naslov je ‘’EE Design’’’

IV. Изрази и оператори

Израз је конструкција која враћа вредност. Састоји се од операнда и оператора. Оператори дефинишу операције које треба извршити над операндима да би се добио резултат. Операција пресликава коначан скуп података (операнда) у коначан скуп података (резултата). Неки оператори се понашају другачије у зависности од типа података са којим раде.

1. Аритметички оператори

Аритметички оператори прихватају као операнде и реалне и целобројне бројеве.
Разликујемо:+ (сабирање), - (одузимање), * (множење), / (дељење), Div (целобројно дељење), Mod (остатак целобројног дељења). Имамо још и унарне промене знака: + (плус) и – (минус).

2. Релациони оператори

Разликујемо: = (једнако), <> (различито), > (веће), < (мање), >= (веће једнако) и <= (мање једнако).

3. Логички BOOLEAN оператори

Разликујемо: **Not** (не), **And** (и), и **Or** (или).
Уз помоћ релацијских и логичких оператора формирају се сложени логички, условни изрази.

p	q	Not p	p Or q	p And q
False	False	True	False	False
False	True	True	True	False
True	False	False	True	False
True	True	False	True	True

4. Редослед важности оператора

Ранг 1	Not						
Ранг 2	*	/	Div	Mod	And		
Ранг 3	+	-	Or				
Ранг 4	=	<>	<	>	<=	>=	In

У случају када оператори имају исти ранг извршавају се са леве на десно. Функције имају виши приоритет, а изрази у заградама највиши.

V. Декларација константи и променљивих

1. Константе

Константа је именована јединица (објекат) програма чија се вредност не мења током извршавања програма. По дефиницији константа се може користити било где у програму уместо стварне вредности. Употреба константи има неколико значајних предности:

- Код је разумљивији и лакши за читање (MAX_BRZINA је разумљивије од 250)
- Програм је ефикаснији, бржи, јер се константе некад уграђују у извршни фајл, или користе део меморије из кога се брже учитава вредност
- Лакша је измена кода. Уколико се нека константна вредност користи на сто места у коду, а дошло је до измене њене вредности, измена кода преко дефинисане константе се обавља на једном месту (месту где смо дефинисали константу). У супротном морали би смо да измену извршимо на свим местима где се појављује константна вредност, што је напорно и мучно.

По конвенцији, идентификатори константи се обично формирају од великих слова. Уколико у имену има више речи, одвајање се изводи преко знака за подвучено (_).

Синтакса: **const** <Identifikator> = <KonstantaIzraz>

Пример: **Const**

```
MAX = 1024;
G = 9.81;
KNJIGA= 'Programiranje';
KRAJ= '.';
```

2. Променљиве

Променљива је именована складишна локација чија вредност се може мењати за време извршавања програма. Декларацијом променљиве одваја се потребан простор у меморији, чија величина зависи од тога ком типу података припада променљива. Прво додељивање вредности променљивој у програму назива се иницијализација променљиве.

3. Локалне и глобалне променљиве

Глобалне (или јавне) променљиве су оне које се декларишу у public (јавном) делу interface секције одређеног унита. Локалне (или приватне) променљиве се декларишу у блоку процедуре. Глобалне и локалне променљиве разликују се по опсегу и животном веку. Опсег одређује видљивост променљиве. Тако су глобалне променљиве видљиве (могу се референцирати) у читавом програма, и постоје све време извршавања програма. Локалне променљиве се виде само унутар процедуре у којој су декларисане, и по завршетку процедуре престају да постоје (не заузимају више меморијски простор).

Пример: **Var** Broj, i : Integer;

```
Kat1, Kat2, Hip : Real;
Znak : Char;
PunoIme : String;
Prosao, Validan: Boolean;
```

VI. Типови података

Тип података је коначан уређен скуп вредности које се могу доделити, и операција које се могу извршити над тим вредностима.

Податке у Pascaly делимо у пет основних типа (класа) података:

1. Целобројни тип (Integer, Shortint, Smallint, Longint, Int64, Byte, Word, LongWord)
2. Реални тип (Real, Real48, Single, Double, Extended, Comp, Currency)
3. Логички тип (Boolean)
4. Знаковни тип (Char)
5. Стрингови (String)
6. Структуирани тип (Array, Record, File, Set)

1. Целобројни типови

Целобројни тип обухвата коначан подскуп целих бројева. Погодни су за коришћење у петљама као бројачи или за индексирања. Генерички целобројни типови су Integer (Longint) и Cardinal (Longword). Треба их користити кад год је то могуће зато што резултују најбољим перформансама за дати процесор и оперативни систем.

Табела: Фундаментални целобројни типови:

Тип	Опсег	Бита
Shortint	-128..127	signed 8-bit
Smallint	-32768..32767	signed 16-bit
Longint	-2147483648..2147483647	signed 32-bit
Int64	-2^63..2^63-1	signed 64-bit
Byte	0..255	unsigned 8-bit
Word	0..65535	unsigned 16-bit
Longword	0..4294967295	unsigned 32-bit

Важе оператори: унарни промена знака: + и -, као и бинарни: +, -, *, Div, Mod.

2. Реални типови података

Реални типови представљају континуалан скуп бројева у децималном запису или са покретним зарезом. Представљају се у позиционом или експоненцијалном запису: +14.234, 0.0, 2.3E-15, -24.345E2. Генерички реални тип је представљен са Real (Double) типом.

Табела: Фундаментални реални типови:

Тип	Опсег	Зн. цифре	Бајта
Real48	2.9e-39..1.7e38	11-12	6
Single	1.5e-45..3.4e38	7-8	4
Double	5.0e-324..1.7e308	15-16	8
Extended	3.6e-4951..1.1e4932	19-20	10
Comp	-9.2e18..9.2e18	19-20	8
Currency	-922337203685477.5808.. 922337203685477.5807	19-20	8

Пошто су приближни не могу се користити у петљама, индексима и сл. На реалном типу дефинисани су стандардни оператори: унарни промене знака: + и -, као и бинарни: +, -, *, /.

3. Логички тип (Boolean)

Ово је такође стандардни, предефинисани тип и може да има само две вредности: True и False. Пример:

var Istina: Boolean;

4. Знаковни тип (Char)

Char је стандардни, унапред дефинисани редни тип података и представља линеарно уређени коначан скуп знакова (слова, бројеви, посебни симболи). Сваком знаку придружен је редни број (његов интерни код). Дефинисан је сет од 256 карактера – ANSI сет где је првих 128 знака припада тзв. ASCII (American Standard Code for Information Interchange) сету. Заузима 1 бајт.

Декларација: **var** Slovo1, Slovo2: Char;

5. Стринг тип (String)

Променљива стринг типа представља секвенцу карактера динамичке дужине, максималне величине између 1 и 255.

Пример:

```
var Tekst: String[50];        {Maksimum 50 karaktera}
    MaksTekst: String;        {Do 255 karaktera}
```

Када се у декларацији не наведе дужина између средњих заграда, онда се одваја локација за максималну дужину стринга od 255 karaktera. Стринг тип се може исказати преко знаковног типа Char: **Array**[1..255] **Of** Char;

Текст се при додели уоквирује са једноструким наводницима Tekst:= ‘Neki tekst!’. При штампању текст наравно нема наводнике (биће Neki tekst!). Уколико треба да се виде наводници онда се користи Tekst:= ‘’’NekiTekst!’’’ што даје при штампању ‘NekiTekst!’.

Стрингови дозвољавају само операције упоређивања, па важе релацијски оператори: =, <>, >, <, >=, <=, док оператор сабирања + има улогу спајања. Када се врши поређење, операција се врши карактер по карактер и зависи од позиције карактера у ASCII табели. Тако је стринг ‘MISAO’ не само различит од стринга ‘misao’, већ је и мањи (‘M’ < ‘м’).

VII. Контрола тока програма, подела наредба

<**Наредба**> ::= <Проста наредба> | <Сложена наредба>

Разликујемо:

- Просте наредбе
- Сложене (структуиране) наредбе

1. Просте наредбе

<**Проста наредба**>::=<Наредба доделе> | <Позив процедуре> | <Goto наредба> | <Празна наредба>

А. Наредба доделе

<**Наредба доделе**> ::= {<Променљива>**:=** <Израз>} | {<Идентификатор функције>**:=** <Израз>}

Пример: Broj := 1000;
Broj := Sqr(i);

Б. Позив процедуре

<**Позив процедуре**> ::= <Идентификатор процедуре> |
<Идентификатор процедуре> **(** <Стварни параметри> [**,**<Стварни параметри>]**…)**

Пример: WriteLn;
WriteLn(Broj, ‘ Hi!’);

В. Goto наредба

<**Goto наредба**> ::= Goto <Лабела>

Пример: Goto 9999;

Г. Празна наредба

<**Празна наредба**> ::= <Празно>
<**Празно**> ::=

2. Сложене – структуиране наредбе

<**Сложена наредба**> ::= <Саставна блок наредба> | <Условна наредба> | <Понављајућа наредба> | <With наредба>

А. Саставна – блок наредба

<**Саставна блок наредба**> ::= **Begin** <Наредба> [**;**<Наредба>]**… End**

Саставна наредба чини низ наредба које су одвојене са ‘ ; ‘ и уоквирене кључним речима Begin и End. Програм у Pascaly можемо посматрати као једну сложену наредбу.

Пример:
```
Begin
    Write(‘Upisi broj: ’);
    Readln(Broj);
    Writeln(‘Broj je: ‘,Broj:6)
End;
```

Б. Условне наредбе – IF

<**Условна наредба**> ::= <If наредба> | <Case наредба>
<**If наредба**> ::= **If** <Израз> **Then** <Наредба> [**Else** <Наредба>]

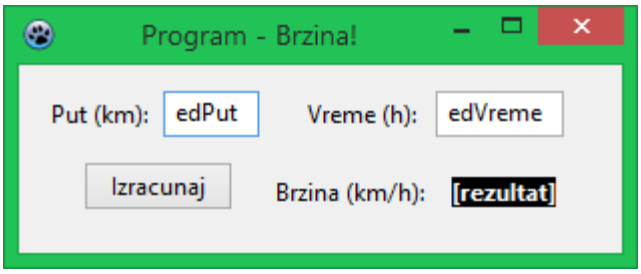
Код If наредбе имамо логички израз који мора да еволуира или у True када се извршава једна грана наредбе или False када се извршава друга грана наредбе (или ниједна).
У том смислу имамо два облика if наредбе: **if**...**then** и **if**...**then**...**else**.

Синтакса **if**...**then** форме:

If <Израз> **Then** <Наредба>
Ако је израз True наредба ће бити извршена, у супротном неће.

Пример:

```
 1:    procedure Tform1.BtIzracunajClick(Sender: Tobject);
 2:      var Put, Vreme, Brzina: real;
 3:    begin
 4:      Put:= StrToFloat(edPut.Text);
 5:      Vreme:= StrToFloat(edVreme.Text);
 6:      Brzina:= 0;
 7:      if Vreme > 0 then
 8:        Brzina:= Put / Vreme;
 9:      lbBrzina.Caption:=FloatToStr(Brzina);
10:    end;
```




```
 7:    if Vreme > 0 then
 8:      begin
 9:        Brzina:= Put / Vreme;
10:        lbBrzina.Caption:=FloatToStr(Brzina);
11:      end;
```

Синтакса **if**...**then**...**else** форме**:**

If <Израз> **Then** <Наредба1> **Else** <Наредба2>

Ако је израз True, онда се извршава наредба1 у супротном наредба 2. На пример:

```
 7:    if Vreme > 0 then
 8:      begin
 9:        Brzina:= Put / Vreme;
10:        lbBrzina.Caption:=FloatToStr(Brzina);
11:      end
12:    else
13:      Exit;
```

И then и else клаузуле садрже по једну наредбу свака, али та наредба може такође бити структуирана наредба, нпр.:

```
 7:     if Vreme > 0 then
 8:       begin
 9:          Brzina:= Put / Vreme;bmnb
10:
11:          lbBrzina.Caption:=FloatToStr(Brzina);
12:       end
13:     else
14:        begin
15:           lbBrzina.Caption:= 'Nedozvoljena brzina!'
16:           Exit;
17:       end;
```

Обратите пажњу да тачка зарез се никад не поставља између then клаузуле и речи else.

Case наредба

Case наредба пружа читљивију алтернативу сложеним дубоко угњежденим If наредбама.
Њен облик је следећи:

```
case <Selektor Izraz> of
   <CaseLista1> : <Naredba1>;
   <CaseLista2> : <Naredba2>;
    ...
   <CaseListaN> : <NaredbaN>;
end
```

где **Selektor Izraz** може бити било који израз редног типа (integer, char) и свака CaseLista састављена од бројева, константи или израза компатибилних са Selektor Izraz-ом.

Може бити у форми листе: Vrednost1, Vrednost2, …, VrednostN или
У форми опсега: Prvi…Poslednji где је Prvi мањи или једнак Poslednjem.

Case наредба може имати и финалну **else** клаузулу:

```
case <Selektor Izraz> of
        <CaseLista1> : <Naredba1>;
    <CaseLista2> : <Naredba2>;
        ...
        <CaseListaN> : <NaredbaN>;
   else
    <Naredbe>;
end
```

Пример:

```
case I of
  1..5: Rezultat := 'Low';
  6..9: Rezultat := 'High';
  0, 10..99: Rezultat := 'Out of range';
else
  Rezultat := '';
end;
```

је еквивалентно угњежденим if наредбама:

```
if I in [1..5] then
  Rezultat := 'Low'
  else if I in [6..10] then
    Rezultat := 'High'
    else if (I = 0) or (I in [10..99]) then
      Rezultat := 'Out of range'
      else
        Rezultat := '';
```

B. ПОНАВЉАЈУЋЕ НАРЕДБЕ – СТРУКТУРЕ ЦИКЛУСА

<**Понављајућа наредба**> ::= <For наредба> | <While наредба> | <Repeat наредба>
<**For наредба**> ::= **For** <Контролна променљива> **:=** <Почетна вредност> **{To** | **Downto}** <Финална вредност> **Do** <Наредба>
<**While наредба**> ::= **While** <Израз> **Do** <Наредба>
<**Repeat наредба**> ::= **Repeat** <Наредба> [**;**<Наредба>]**... Until** <Израз>
<**With наредба**> ::= **With** <Record листа променљивих> **Do** <Наредба>
<**Record листа променљивих**> ::= <Record променљива> [**,**<Record променљива>]**...**

For наредба се користи у случајевима када тачно знамо колико пута ће се циклус одиграти. Контролна променљива мора да буде редног типа (byte, integer, longint, набројани...). Финална и крајња вредност могу бити и изрази. Начин функционисања For наредбе:
Корак 1: на почетку циклуса, почетна вредност се додељује контролној променљивој.
Корак 2: проверава се да ли је контролна променљива већа од финалне вредности (ако је To облик), или мања (ако је Downto облик).
Корак 3: ако је то истина For наредба се завршава, а у супротном се извршавају наредбе у For блоку.
Корак 4: контролна променљива се повећава за 1 (To облик), или смањује за 1 (Downto облик) и извршавање се враћа на врх петље, где се поново извршава корак 2.

While и Repeat наредбе се користе у случајевима када је непознат број поновљања. While је наредба са предусловом, а Repeat са постусловом.

У While структури наредба (наредбе) се извршава све док је логички израз истинит (True). У Repeat структури наредба се извршава све док се не испуни услов (док не постане True). Обе структуре се користе у случајевима када није познат тачан број понављања. Разлика је што се у While структури одмах проверава услов, и ако је он у старту False нећемо имати ни један циклус. Repeat структура са друге стране прво извршава наредбу, па затим проверава услов, и тако се мора одиграти бар један циклус.се у While структури одмах проверава услов, и ако је он у старту False нећемо имати ни један циклус. Repeat структура са друге стране прво извршава наредбу, па затим проверава услов, и тако се овде мора одиграти бар један циклус.

Пример: Сума природних бројева, мањих од 100:

```
Program SumaFor;
Var Suma, Brojac: Integer;
Begin
   Suma:=0;
   For Brojac:=1 To 99 Do
      Suma:= Suma+Brojac;
   Writeln (‘Suma je: ‘, Suma)
End.
```

```
Program SumaWhile;
Var Suma, i : Integer;
Begin
   Suma:= 0;
   i:=0;
   While i <100 Do
      Begin
         i:=i+1
         Suma:= Suma+i
      End;
   Writeln(‘Suma je: ‘, Suma)
End.
```

```
Program SumaRepeat;
Var Suma, i : Integer;
Begin
   Suma:= 0;
   i:=1;
   Repeat
      Suma:= Suma+i;
      i:=i+1
   Until i=100;
   Writeln(‘Suma je:‘, Suma)
End.
```

VIII. Процедуре и функције

Процедуре и функције, заједнички названи рутине су самостални блокови наредби или потпрограми које могу бити позиване и коришћене са различитих локација програма.
Функција је рутина која при извршавању враћа вредност, док процедура ради нешто али не враћа вредност.
Позиви функција, пошто враћају вредност, могу да се користе у наредбама доделе и операцијама.
На пример: L:= NekaFunkcija(X); позива NekaFunkcija и додељује резултат променљивој L. Позиви функција не могу да буду на левој страни наредбе доделе. Процедуре и функције могу да позивају рекурзивно саме себе.

Декларација процедура и функција

При декларацији процедура и функција потребно је дати у заглављу име процедуре или функције, тип сваког параметра и у случају функција тип вредности или резултата који ће функција да врати.
Затим се пише у телу процедуре или функције блок кода који се извршава позивом.

1. Декларација функција

Синтакса:

```
function ImeFunkcije(ListaParametara): TipRezultata;
  LokalneDeklaracije (konstante i promenljive const/var);
  begin
    Naredbe
  end;
```

где је ImeProcedure валидан идентификатор, , TipRezultata је идентификатор типа, наредбе су секвенца команди која се извршавају приликом позива, а локална декларација је опциона.

Унутар блока наредби можемо да користимо променљиве и друге идентификаторе декларисане у делу локалне декларације, имена параметара из листе параметара као и све остале идентификаторе унутар опсега где је декларисана сама функција. И само име функције се понаша као специјална променљива која прихвата вредност резултата функције.

2. Декларација процедуре

Има следећи облик:

```
procedure ImeProcedure(ListaParametara);
  LokalneDeklaracije (konstante i promenljive const/var);
  begin
    Naredbe
  end;
```

где је ImeProcedure валидан идентификатор, наредбе су секвенца команди која се извршавају приликом позива, а локална декларација је опциона.

Семантика параметра

Параметри се категорисане на више начина:
Сваки параметар је класификован као вредносни, променљиви, константа или излазни.
Вредносни су подразумевани. Кључне речи var, const, и out дефинишу променљиве, константе или излазне параметре респективно.

Вредносни и променљиви параметри

Већина параметара су или вредносни (подразумевани) или променљиви (var) параметри. Вредносни параметри су пребачени по вредности док су променљиви пребачени референцом.
Погледајмо следећи пример:

```
function DoubleByValue(X: Integer): Integer;     // X is a value parameter
begin
  X := X * 2;
  Result := X;
end;
function DoubleByRef(var X: Integer): Integer;   // X is a variable parameter
begin
  X := X * 2;
  Result := X;
end;
```

Обе функције враћају исти резултат, али само друга DoubleByRef мења вредност променљиве која је пребачена као параметар. На пример:

```
var
  I, J, V, W: Integer;
begin
  I := 4;
  V := 4;
  J := DoubleByValue(I);   // J = 8, I = 4
  W := DoubleByRef(V);     // W = 8, V = 8
end;
```

После извршавања овог кода променљива I, која се пребацује DoubleByValue, има исту иницијално додељену вредност, док променљива V, која је пребачена DoubleByRef, има различиту вредност.
Вредносни параметар делује као локална променљива која се иницијализује на вредност која је пребачена позивом процедуре или функције. Процедура или функција прави њену копију, и све промене се дешавају на копији, док оригинална променљива остаје непромењена.
Променљиви параметар пре делује као показивач него копија. Промене унутар тела процедуре или функције остаје и после враћања извршавања месту позива.

Подразумевани параметри

Можемо да прецизирамо подразумевану вредност параметра у заглављу процедуре или функције тако што ћемо завршити декларацију параметра са симболом = праћеног константним изразом компатибилним са типом параметра.
На пример:

```
procedure UradiNesto(Broj: double; Vrednost: Integer = 0);
```

Следећи позиви процедуре су идентични:

```
UradiNesto(NekiBroj);
UradiNesto(NekiBroj, 0);
```

Параметри са подразумеваном вредношћу морају да се појаве на крају листе параематара.

Позиви процедура и функција

Кад се позивају процедуре или функције, програм пребацује контролу са места одакле је позив направљен на тело рутине. Ако је рутина декларисана са параметрима, у позиву мора да се пребаце параметри који одговарају редоследу и типу листе параметара рутине.
Параметри који се пребацују рутини се називају правим параметрима, док су параметри у декларацији рутине називају формалним параметрима.