Basic	Pascal	c++	Značenje
	Uvod		
REM komentar ' komentar	{ komentar } (* komentar *)	// komentar /* komentar */	komentar (može biti u jednom redu ili između zagrada)
CLS	program imePrograma;	#include <iostream> using namespace std;	početak programa

od početka do kraja kako pišu naredbe	begin
	naredbe
	end.
		int main()
		{
		naredbe
		}
			glavni (izvršni) dio programa

u Basic-u ako želimo više naredbi ispisati u jednom redu koristimo : dvotačku (npr. naredba1: naredba2: naredba3)			
PRINT varijabla;	write(varijabla);	cout<<varijabla;	ispis varijable sa položajem kursora iza nje
PRINT "tekst"	writeln('tekst');	cout<<"tekst"<<endl;	ispis teksta sa položajem kursora u novom redu
; ,	,	<<	znakovi koji nam omogućavaju da u naredbi pisanja povezujemo kombinacije različitih varijabli i drugačijeg teksta
+ - *	+ - *	+ - *	sabiranje, oduzimanje i množenje
u Pascal-u treba obratiti pažnju kod ispisa realnog broja. Naime ako ne stavimo format ispisa (koji ima oblik da se iza ispisujućeg realnog broja stavi :brojC :brojD), ispisuje se broj u eksponencijalnoj notaciji. Dakle rezultat treba ispisati u obliku: *rezultat:brojC:brojD* gdje brojC predstavlja broj cijelih a brojD broj decimalnih mijesta. Ako se brojC prekorači on se zanemaruje		/* u c++ treba voditi računa da bar jedan od dva djeljena broja bude realan broj. U protivnom, kao rezultat djeljenja dva cijela broja, ćemo dobiti cijeli broj (biće izvršeno cjelobrojno djeljenje). Dva cijela broja je moguće podijeliti da kao rezultat dobijemo realan broj i tu je potrebno samo iza jednog broja staviti tačku tj.: *brojA. / brojB* */	
brojA / brojB	brojA / brojB rezultat:brojC:brojD	brojA / brojB brojA. / brojB	djeljenje dva broja
	Promjenjive		
sam Basic ne zahtijeva da se promjenjive prethodno najave ali pri pisanju dužih programa to dovodi do velikih opasnosti od mogućih greški. Ako želimo da deklarišemo varijable, to je potrbno uraditi prije neko što je upotrijebimo u uprogramu	program ... ; uses ... ; label ... ; const ... ; type ... ; var ... ; procedure ... ; function ... ; begin naredbe end.	u c++ jedino je ograničenje da se promjenjiva mora deklarisati prije nego se upotrijebi	generalni izgled čitavog programa u Pascal-u (kojim bi se redom deklaracije trebale primjenjivati)
DIM imeVarijable AS tip	var imeVarijable: tip;	tip imeVarijable;	deklaracija varijable
DIM imeV1 AS tip, imeV2 AS tip	var imeV1, imeV2: tip;	tip imeV1, imeV2;	deklaracija dvije varijable
DIM imeV1 AS tip1, imeV2 AS tip2, imeV3 AS tip3	var imeV1: tip1; imeV2, imeV3: tip2;	tip1 imeV1; tip2 imeV2, imeV3;	deklaracija tri varijable koje mogu biti različitog tipa
INTEGER SINGLE STRING LONG DOUBLE	integer real string longint double char	int float long double char	tipovi varijabli
{ za korištenje tipa double u Pascal-u se na samom početku programa (prije naredbe program) mora dodati: {$N+} }			
INPUT varijabla INPUT "tekst"; varijabla	read(varijabla); readln(varijabla);	cin>>varijabla;	naredba čitanja (u jednom ili dva oblika)
,	,	>>	znakovi koji nam omogućavaju da u naredbi čitanja povezujemo više varijabli
imeVarijable = izraz	imeVarijable := izraz;	imeVarijable = izraz;	znak dodjeljivanja
i=i+j k=k+1 i=j a=a-b j=j+1	i:=i+j; k:=k+1; i:=j; a:=a-b; j:=j+1;	i+=j; k++; i=j++; a-=b; ++k;	ekvivalenti i upotreba znaka dodjele povećavanje i smanjivanje vrijednosti varijable k za jedan
c=c*d k=k-1 j=j+1 e=e/f i=j	c:=c*d; k:=k-1; j:=j+1; e:=e/f; i:=j;	c*=d; k--; i=++j; e/=f; --k;	ekvivalenti i slične upotrebe znakova dodjele
	Složenije računske operacije		
	/* za korištenje matematičkih funkcija u c++ treba, na samom početku programa, uključiti math.h biblioteku, tj. treba napisat: #include <math.h> */		
x ^ y	exp(y*ln(x))	pow(x,y)	x^y (x na y)
EXP(n)	exp(n)	exp(n)	e^n (e na n) e=2.718281828459045240
x ^ 2	sqr(x)	pow(x,2)	x^2 (x na kvadrat)
SQR(m)	sqrt(m)	sqrt(m)	drugi korijen broja m
LOG(a)	ln(a)	log(a)	logaritam broja a po bazi e
LOG(x) / LOG(10)	ln(x) / ln(10)	log10(x)	logaritam broja x po bazi 10
SIN(ugla) COS(ugla) TAN(ugla) ATN(ugla)	sin(ugla) cos(ugla) arctan(ugla)	sin(ugla) asin(ugla) cos(ugla) acos(ugla) tan(ugla) atan(ugla)	trigonometriske funkcije (sinus, kosinus, tangens, arkussinus, arkuskosinus i arkustangens) koje daju ispravan rezultate za ugao u radijanima
ABS(a)	abs(a)	abs(a)	apsolutna vrijednost broja a
INT(x)	trunc(x)	int(x)	odsjeca decimalni dio broja x

ATN(1)*4
radijan=(ATN(1)*4) / 180
ugao=ugao*radijan
	arctan(1)*4
	radijan:=(arctan(1)*4) / 180;
	ugao:=ugao*radijan;
		atan(1)*4
		radijan=(atan(1)*4) / 180;
		ugao*=radijan;
			vrijednost broja pi (pi=3.141592653589793240)
			vrijednost jednog radijana (1rad = pi / 180)
			pretvaranje ugla koji je u stepenima u ugao radijana (da bi se mogle ispravno primjeniti trigonometrijske funkcije)

\ mod	div mod	/ %	računske operacije koje se upotrebljavaju samo na cijelim br.: cijeli dio djeljenja dva broja (cjelobrojno djeljenje) ostatak pri djeljenju (modul)
	Ciklične strukture (petlje)		
Naredbe koje su skoro izašle iz upotrebe: naredba goto i naredba case (naredba goto se treba izbjegavati pod svaku cijenu)			
... ... GOTO labela labela: naredba ... 'ili ... labela: naredba GOTO labela	label imeLabele; label imeLabele; ... ... goto imeLabele; {ili} imeLabele: naredba ... ... imeLabele: naredba goto imeLabele;	... ... goto imeLabele; imeLabele: naredba ... /*ili*/ ... imeLabele: naredba goto imeLabele;	dva načina upotrebe naredbe goto

SELECT CASE varijabla
CASE konstanta
naredbe
...
CASE konstanta
naredbe
CASE ELSE
naredbe
END SELECT
	case varijabla of
	konstanta: naredba;
	...
	konstanta: naredba;
	else
	naredba
	end
		switch(varijabla)
		{
		case konstanta:
		naredba
		...
		}
			opšti oblici case naredbe

Ostale naredbe ponavljanja			
= <> > < >= <=	= <> > < >= <=	== != > < >= <=	relacioni operatori (znakovi jednakosti, različito, veće od, manje od, veće ili jednako i manje ili jednako)

	begin
	...
	end;
		{
		...
		}
			početak i kraj neke cjeline ili bloka (može i tijela petlje)

DO
...
LOOP WHILE uvjet DO
...
LOOP UNTIL uvjet
	repeat
	...
	until uvjet;
		do
		{
		...
		}
		while (uvjet);
			naredba do (tačnije strukture do-loop while i do-while) za Basic i c++
			i naredba repeat-until za Pascal (struktura do-loop until za Basic)

DO WHILE uvjet DO UNTIL uvjet ... ... LOOP LOOP			do petlje kod kojih se uvjet provjerava na početku petlje
EXIT	break;	break;	naredba za izlaženje iz petlji ponavljanja

WHILE uvjet
...
WEND
	while uvjet do
	begin
	...
	end;
		while(uvjet)
		{
		...
		}
			naredba while

FOR varijablaP=prvaVrijednost TO broj
naredbe
NEXT varijablaP
 	
FOR varijablaP=nekiBrojA TO NekiBrojB STEP brojK
naredbe
NEXT VarijablaP
	for i:=prviBroj to broj do
	begin
	naredbe
	end;
	for brojač:=zadnjeBroj downto nekiBroj do
	begin
	naredbe
	end;
		for (inicijalizacija; uvjet; izmjena)
		{
		tijeloPetlje
		}
		
		for (i=10; i>=1; i--)
		cout<<i<<endl;
		/* gornji primjer ispisuje brojeve od 10 do 1 unatraške */
			neki od oblika naredbe for

	Strukture grananja		

IF uvjet THEN
naredbe
ELSE
naredbe
END IF
	if uvjet then
	begin
	naredbe
	end
	else
	naredba
		if (uvjet)
		naredba
		else
		{
		naredbe
		}
			naredba grananja if

RANDOMIZE TIMER
a=INT(1000*RND)
'RND se ponaša kao promjenjiva čija je vrijednost veća ili
'jednaka nuli a strogo manja od jedinice (0<=RND<1)
	randomize;
	a:=trunc(1000*random);
	
	{ riječ random ima isto značenje kao riječ RND u Basic-u }
		randomize();
		a=random(1000);
		/* za korištenje funkcije random() potrebno je uključiti stdlib.h biblioteku, tj. treba napisati: #include <stdlib.h> */
			uključivanje generatora slučajnih brojeva
			broj a će dobiti vrijednost iz intervala {0,1,...,999}

AND OR NOT(izraz)	and or not(izraz)	&& || !(izraz)	konjukcija, disjunkcija i negacija
	Nizovi kao složene strukture podataka		

DIM imeNiza (brojElemenata) AS tip		tip *imeNiza;
		...
		//kad unesemo broj elemenata
		imeNiza = new tip[brojElemenata+1];
			dinamički način definiranja niza

	var imeNiza: array[1..brojElemenata] of tip;	tip imeNiza[brojElemenata+1];	statički način definiranja niza
naredba imeNiza(brojElementa)	naredba imeNiza[brojElementa]	naredba imeNiza[brojElementa]	pristup nizu

DIM imeMatrice(brojRedova, brojKolona) AS tip naredba imeMatrice(brojReda,brojKolone)	var imeMatrice: array[1..brojRedova, 1..brojKolona] of tip; naredba imeMatrice[red,kolona]	tip imeMatrice[brojRedova+1][brojKolona+1]; naredba imeMatrice[red][kolona]	definiranje matrice pristup matrici
	Stringovi (alfanumeričke promjenjive)		
DIM imeS AS STRING	var imeS: string;	char imeS[maksimalnaDužinaStringa];	definiranje alfanumeričke promjenjive pod imenom: imeS
imeS="tekst"	imeS:='tekst';		dodjeljivanje vrijednosti alfanumeričkoj promjenjivoj
+	+		znak za nadovezivanje stringova
INPUT imeS	readln(imeS);	cin.getline(imeS,maksimalnaDužinaStringa); cin>>imeS;	učitavanje stringova u varijablu imeS učitavanje stringova u varijablu imeS do prvog praznog mjesta
LEN(imeS)	length(imeS)	strlen(imeS) /* za korištenje funkcije strlen() potrebno je uključiti string.h biblioteku, tj. treba napisati: #include <string.h> */	funkcija koja za rezultat vraća broj znakova alfa. promj. imeS
LEFT$(imeS,m)			izdvaja prvih m znakova iz alfa. promjenjive imeS
RIGHT$(imeS,m)			izdvaja posljednjih m znakova iz alfa. promjenjive imeS
MID$(imeS,m,n)	copy(imeS,m,n)		izdvaja n znakova iz prom. imeS počevši od m-tog znaka
MID$(imeS,i,1)	imeS[i]	imeS[i]	pristup i-tom znaku u stringu imeS
MID$(imeS,m)			izdvaja sve znakove iz imeS počevši od m-tog znaka do kraja
	Potprogrami		
... GOSUB linija ... END linija: naredba ... RETURN			poziv i pisanje potprograma u izvornom BASICU (u ovom slučaju potprogram smo napisali iza glavnog programa mada se u načelu mogu napisati bilo gdje)
'potprogrami se u QBasicu počnu pisati u glavnom prozoru s tim što čim ukucamo zaglavlje 'potprograma QBasic nam automatski otvori novi potpuno prazan prozor rezervisan samo za taj 'potprogram. Kasnije se sa pritiskom na tipku F2 možemo prebacivati između glavnog programa 'i potprograma pomoću menija koji se tom prilikom prikaže na ekranu	{potprogrami se u Pascal-u pišu prije početka glavnog programa}	/*u c++ potprogrami se mogu pisati bilo prije bilo poslije glavnog programa. Ja ću, radi nešto lakše varijante, potprograme pisati prije početka glavnog dijela programa*/	

1^{O} SUB imePotprograma
naredbe
END SUB
	1^{O} procedure imePotprograma;
	...
	begin
	naredbe
	end;
		1^{O} void imePotprograma()
		{
		naredbe
		}
			definiranje potprograma

'potprograme u Basic-u možemo pozivati na dva načina: pomoću korištenja naredbe CALL ili kucajući samo ime potprograma. U drugom slučaju na samom početku glavnog programa je potrebno 'dodati naredbu DECLARE iza koje slijedi zaglavlje potprograma (QBasic nam ovdje pruža nekakvu prednost jer čim snimimo program on će automatski dodati naredbu DECLARE)			
CALL imePotprograma DECLARE SUB imePotprograma() . . . imePotprograma	imePotprograma;	imePotprograma();	dva načina pozivanja potprograma 1^{O} za Basic i poziv potprograma 1^{O} za Pascal i c++

2^{O} SUB imePotprograma
DIM imeVarijable AS tip
...
END SUB
	2^{O} procedure imePotprograma;
	var imeVarijable: tip;
	...
	begin
	naredbe
	end;
		2^{O} void imePotprograma()
		{
		tip imeVarijable;
		...
		}
			definiranje potprograma koji će koristiti svoju lokalnu promjenjivu

CALL imePotprograma DECLARE SUB imePotprograma() . . . imePotprograma	imePotprograma;	imePotprograma();	dva načina pozivanja potprograma 2^{O} za Basic i poziv potprograma 2^{O} za Pascal i c++
'u Basicu ako želimo da istu varijablu možemo koristiti u dva ili više potprograma (da ta varijabla bude 'zajednička), prvo je potrebno da tu varijablu u glavnom programu definišemo sa *DIM imeVarijable AS tip* pa 'onda u potprogramu u kojem je želimo koristiti, dodamo na početku *SHARED imeVarijable AS istiTip*	sasvim je legalno da postoje lokalne i globalne promjenjive istog imena. Tada unutar potprograma vrijedi lokalna promjenjiva a ne globalna promjenjiva istog imena, dok u ostatku programa vrijedi globalna promjenjiva		

3^{O} SUB imePotprograma(formalniParametar AS tip)
...
END SUB
	3^{O} procedure imePotprograma(formalniParametar:tip);
	...
	begin
	naredbe
	end;
		3^{O} void imePotprograma(tip formalniParametar)
		{
		naredbe
		}
			definiranje potprograma koji će koristiti prenos parametara po vrijednostima (vrijednosni parametri) u Pascal-u i c++-u
			i
			definiranje potprograma sa jednim formalnim parametrom u Basic-u

CALL imePotprograma(stvarniParametar) DECLARE SUB imePotprograma(formalniParametar AS tip) . . . imePotprograma stvarniParametar	imePotprograma(stvarniParametar);	imePotprograma(stvarniParametar);	dva načina pozivanja potprograma 3^{O} za Basic (ukoliko je stvarni parametar promjenjiva prenos se vrši po referenci) i poziv potprograma 3^{O} za Pascal i c++
CALL imePotprograma((stvarniParametar)) DECLARE SUB imePotprograma(formalniParametar AS tip) . . . imePotprograma (stvarniParametar)	CALL imePotprograma(stvarniParametar+0) DECLARE SUB imePotprograma(formalniParametar AS tip) . . . imePotprograma 0+stvarniParametar		neki od načina pozivanja potprograma 3^{O} za Basic pri čemu će stvarni parametar biti prenesen po vrijednosti

4^{O} SUB imePot(forPar1 AS tip, foPar2 AS tip, forPa3 AS tip)
...
END SUB
	4^{O} procedure imePotpr(forPar1, forPar2, forPar3 :tip);
	...
	begin
	naredbe
	end;
		4^{O} void imePotpr(tip forPar1, tip forPar2, tip forPar3)
		{
		naredbe
		}
			definiranje potprograma koji koristi tri formalna parametra i koji će koristiti prenos parametara po vrijednostima (vrijednosni parametri) u Pascal-u i c++-u
			i definiranje potprograma koji koristi tri formalna parametra u Basic-u

CALL imePotpr(stvPar1, stvPar2, stvPar3) DECLARE SUB imPo(foP1 AS tip, foP2 AS tip, foP3 AS tip) . . . imePotprograma stvPar1, stvPar2, stvPar3	imePotprograma(stvPar1, stvPar2, stvPar3);	imePotprograma(stvPar1, stvPar2, stvPar3);	dva načina pozivanja potprograma 4^{O} za Basic (ukoliko je stvarni parametar promjenjiva prenos se vrši po referenci) i poziv potprograma 4^{O} za Pascal i c++
CALL imePotpr((stvPar1), (stvPar2), (stvPar3)) DECLARE SUB imPo(foP1 AS tip, foP2 AS tip, foP3 AS tip) . . . imePotpr(stvPar1), (stvPar2), (stvPar3)	CALL imePotpr(stvPar1+0, stvPar2+0, stvPar3+0) DECLARE SUB imPo(foP1 AS tip, foP2 AS tip, foP3 AS tip) . . . imePotpr stvPar1+0, stvPar2+0, stvPar3+0		neki od načina pozivanja potprograma 4^{O} za Basic pri čemu će stvarni parametar biti prenesen po vrijednosti
' pri definisanju potprograma kao f-ja koja će vratiti vrijednost treba definisati kojeg će tipa f-ja biti. U Basic-u to je moguće samo 'ako se iza imena f-je i u deklaraciji i u definiciji f-je doda neki specijalni znak. Sledeći znakovi predstavljaju tipove funkcije: **%** (INTEGER), **&** (LONG), **!**(SINGLE), **#**(DOUBLE), **$**(STRING)		formalni parametri su lokalne promjenjive, dok stvarni parametar može biti neka vrijednost, varijabla(kojoj je dodjeljena vrijednost) ili neki izraz	

5^{O} FUNCTION imePotprZnakTipaFje(formalParame AS tip)
...
imePotprograma=traženaVrijednost
END SUB
	5^{O} function imePotprograma(formalniParametar:tip):tip;
	...
	begin
	...
	imePotprograma:=traženaVrijednost;
	end;
		5^{O} tip imePotprograma(tip formalniParametar)
		{
		...
		return traženaVrijednost;
		}
			definiranje potprograma kao funkciju (funkcija je potprogram koja za rezultat vraća neku vrijednost koja nam je potrebna za dalji rad)

DECLARE FUNCTION imePotprZnTiFje(forPar AS tip) . . . naredba imePotprograma(stvarniParametar)	naredba imePotprograma(stvarniParametar)	naredba imePotprograma(stvarniParametar)	poziv potprograma 5^{O}

6^{O} FUNCTION imePotZTF(forPar1 AS tip, forPar2 AS tip)
...
imePotprograma=traženaVrijednost
END SUB
	6^{O} function imePotprograma(forPar1, forPar2:tip):tip;
	...
	begin
	...
	imePotprograma:=traženaVrijednost;
	end;
		6^{O} tip imePotprograma(tip forPar1, tip forPar2)
		{
		...
		return traženaVrijednost;
		}
			definiranje potprograma kao funkciju koja koristi dva formalna paramerra

DECLARE FUNCTION imPZTF(foP1 AS tip, foP2 AS tip) . . . naredba imePotprograma(stvPar1, stvPar2)	naredba imePotprograma(stvPar1, stvPar2)	naredba imePotprograma(stvPar1, stvPar2)	poziv potprograma 6^{O}

7^{O} FUNCTION iPoZt(fP1 AS tp1, fP2 AS tp2, fP3 AS tp3)
...
imePotprograma=traženaVrijednost
END SUB
	7^{O} function imePo(foPa1: tip1, foPa2:tip2, foPa3:tip3):tip;
	...
	begin
	...
	imePotprograma:=traženaVrijednost;
	end;
		7^{O} tip imePotpr(tip1 forPar1, tip2 forPar2, tip3 forPar3)
		{
		...
		return traženaVrijednost;
		}
			generalno potprogrami mogu imati više parametara koji su različitog tipa. Ovo je jedan primjer za definiranje potprograma kao funkcije koja koristi tri formalna parametra, koja mogu biti različitog tipa

DECLARE FUNCTION iZ(fp1 AS t1, fp2 AS t2, fp3 AS t3) . . . naredba imePotprograma(stvPar1, stvPar2, stvPar3)	naredba imePotprograma(stvPar1, stvPar2, stvPar3)	naredba imePotprograma(stvPar1, stvPar2, stvPar3)	poziv potprograma 7^{O}

8^{O} DEF FNimeFunkcije(forPar1,..., forParN)=izraz	8^{O} procedure imePotprograma(var formalniParametar:tip);
	...
	begin
	naredbe
	end;
		8^{O} void imePotprograma(tip &formalniParametar)
		{
		naredbe
		}
			definiranje potprograma koji će koristiti prenos parametara po referenci (promjenjivi parametri) u Pascal-u i c++-u
			i
			definiranje funkcijeje za Basic u jednoj liniji koda

naredba FNimeFunkcije(stvPar1,...,stvParN)	imePotprograma(stvarniParametar);	imePotprograma(stvarniParametar);	poziv potprograma 8^{O}

	9^{O} procedure imePotpr(var forPar1, forPar2, forPar3 :tip);
	...
	begin
	naredbe
	end;
		9^{O} void imePotpr(tip &forPar1, tip &forPar2, tip &forPar3)
		{
		naredbe
		}
			definiranje potprograma koji koristi tri formalna parametra i koji će koristiti prenos parametara po referenci (promjenjivi parametri) u Pascal-u i c++-u

	imePotprograma(stvPar1, stvPar2, stvPar3);	imePotprograma(stvPar1, stvPar2, stvPar3);	poziv potprograma 9^{O}
prenos parametara po vrijednosti podrazumjeva da se stvarni parametar ne može promjeniti. Parametri koji se prenose po vrijednosti nazivaju se vrijednosnim parametrima prenos parametara po referenci podrazumjeva da će stvarni parametar dobiti vrijednost koju želimo i čiju izmjenu ćemo definisati unutar nekog potprograma. U ovom slučaju stvarni parametar mora biti neka varijabla. Parametri koji se prenose po referenci nazivamo promjenjivi parametrima			
potprograme možemo učiniti praktički neovisnim od glavnog programa. Ovo je upravo jedno od osnovnih zadataka metodologije programiranja poznate pod nazivom struktuirano ili modularno programiranje. Pod tim podrazumjevamo razbijanje programa na više manjih cjelina - modula koji se mogu razvijati neovisno jedan od drugoga			
programske jezike možemo podjeliti na: jezike niskog nivoa 1.mašinski jezik(kombinacija 0 i 1), 2.asembler (ili simbolički jezik) i jezike visokog nivoa 3.proceduralni jezici 4.neproceduralni jezici. Za proceduralne jezike je neophodno znati definisati algoritam tj. jasan i nedvosmislen niz koraka koja u konačnom nizu dolazi do rešenja ili do zaključka da zadatak nema rješenja. Proceduralni jezici su: FORTRAN, COBOL, od novijih: BASIC, Pascal, C, c++ i tipični predstavnici Objektno orjentisanih jezika (posebna vrsta procedural. jezika): Visual Basic, Object Pascal (Delphi), C++ i Java. Kod neproceduralnih jezika nemora se sastavljati algoritam već moramo jasno definisati od čega se problem sastoji. Koriste se za baze podataka. Predstavnici su: SQL, Prolog, Lisp.			