

UVOD

Programirati začnemo tedaj, ko želimo s pomočjo računalnika reševati kakšne probleme. Pri tem moramo računalniku posredovati točna navodila, kako naj problem rešuje. Računalniški program razvijamo podobno, kot rešujemo poljuben problem. Sestavlja ga pet glavnih korakov:

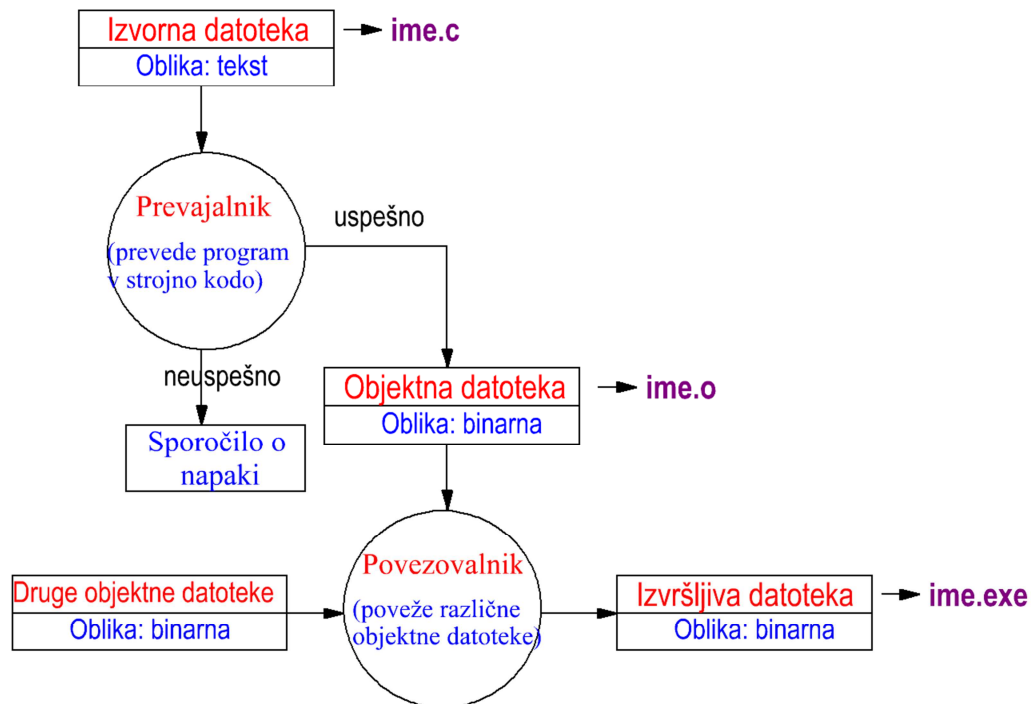
- opredelitev problema (določitev tega, kaj vemo - katere podatke, ki določajo problem, poznamo, in tega, kaj želimo dobiti – rezultat),
- načrtovanje postopka rešitve,
- zapis postopka rešitve v programskem jeziku,
- izvršitev programa na računalniku,
- preverjanje programa (preverjanje pravilnosti rešitve).

Ko začnemo načrtovati rešitev, problem razgradimo na zaporedje korakov, ki pripeljejo do rešitve. Zapis teh navodil imenujemo **algoritem**. Algoritem želimo podati kot zaporedje navodil za računalnik. Za izvajanje navodil v računalniku poskrbi procesor. Torej morajo biti navodila v taki obliki, da jih bo procesor razumel. Zato jih napišemo v kakem programskem jeziku. Vsak programski jezik točno določa besede in pravila, ki se jih pri zapisovanju držimo. Ima enostavno zgradbo in je zato mogoče program, zapisan v programskem jeziku, avtomatsko prevesti v strojni jezik. Poznamo cel kup programskih jezikov. Mi bomo uporabljali programski jezik C.

Če se omejimo na programiranje v ožjem smislu, bomo pod tem razumeli postopek zapisa rešitve problema v izbranem programskem jeziku in izvedbo tega zapisa na računalniku. Pri tem gremo skozi štiri faze. Pri vsaki si pomagamo s programom, ki je že pripravljen na računalniku. Te faze so:

- zapis v programskem jeziku (urejevalnik - *Editor*);
- prevajanje (prevajalnik - *Compiler*);
- povezovanje (povezovalnik - *Linker*);
- izvajanje (*Execute*).

Za programiranje torej potrebujemo program s katerim bomo naredili datoteko z zapisom programa, ter programe s katerim bomo ta zapis prevedli, povezali v ustrezno celoto in izvedli. Za to bomo uporabljali razvojno okolje Dev-C++.



Slika 1: Kako pridemo do izvršljive datoteke

Programski jeziki imajo točno določeno sintakso, ki se je moramo pri programiranju strogo držati. To nam potem omogoča avtomatično prevajanje v strojni jezik. Če so v programu sintaktične napake, jih prevajalnik odkrije in nanje opozori. Prevajalnik pa ne odkrije semantičnih (pomenskih) napak. Za odpravljanje semantičnih napak je potrebno dobro razumevanje problema.

1 PREPROST PROGRAM V C-ju (Izpisovanje nizov)

Program v C-ju je sestavljen iz funkcij (funkcija je del kode, ki nekaj naredi). Program mora vsebovati vsaj eno funkcijo, ki jo imenujemo `main()`. Imenu sledijo parametri v okroglih oklepajih. Programska koda vsake funkcije se nahaja med zavitima oklepajema `{ }`. Programska koda lahko vsebuje več stavkov, ki se izvršijo, ko pokličemo funkcijo. Vsak stavek se konča s podpičjem. Če hočemo izpisati podatke, moramo uporabiti standardno funkcijo `printf`. Funkcije so združene v knjižnice, ki jih moramo na začetku vključiti v program. To storimo z ukazom `#include<ime knjižnice.h>`.

V razvojnem okolju Dev-C++ dobimo predlogo za programiranje na sliki 1, ki jo potem nekoliko spremenimo (slika 2).

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5
6     printf("Press enter to continue ...");
7     getchar();
8     return 0;
9
10 }

```

Slika 2: Dev-C++ predloga za programiranje v C-ju

```

1 #include <stdio.h>
2
3 int main()
4 {
5
6     getchar();
7     return 0;
8 }

```

Slika 3: Spremenjena predloga za programiranje

```

1 /*
2     Author: Franc Kamenik
3     Date: 17.02.13 22:31
4     Description: Izpis pozdrava
5 */
6
7 #include <stdio.h>
8
9 int main()
10 {
11     printf("Pozdravljeni! ");
12     printf("Kako ste?");
13     getchar();
14     return 0;
15 }
16

```

komentar (prevajalnik ga ne upošteva)

← vključimo knjižnico funkcij

telo funkcije main()

Pozdravljeni! Kako ste?_

Slika 4: Prvi program Izpis pozdrava in izpis na zaslonu

Program nize izpisuje drugega za drugim in gre v novo vrstico šele, ko mu zmanjka prostora na zaslonu. V izpis lahko vključimo posebne znake, kot so.

- `\n` - nova vrsta,
- `\t` - tabulator,
- `\"` - narekovaj,...
- `\b` - pomik za eno mesto nazaj (tudi pobriše znak)
- `\f` - znak za novo vrstico (form feed)
- `\n` - znak za novo vrstico (newline)
- `\r` - pomik na začetek vrstice (carriage return)
- `\v` - vertikalni tabulator (vertical tab)
- `\'` - enojni narekovaj (single quote)
- `\\` - znak za `\` (backslash)
- `\0` - znak za konec niza (null character)
- `\o` - osmiška konstanta; octal constant
- `\x` - šestnajstiška konstanta (hexadecimal constant)

```
[*] Pozdrav.c
1  /*
2   Author: Franc Kamenik
3   Date: 17.02.13 23:10
4   Description: Izpis pozdrava drugače
5  */
6
7  #include <stdio.h>
8
9  int main()
10 {
11     printf("Pozdravljeni!\n");
12     printf("Kako ste?\n");
13     getchar();
14     return 0;
15 }
```

```
Pozdravljeni!
Kako ste?
_
```

Slika 5: Koda za Izpis pozdrava drugače in izpis

2 PODATKI

Program dela s podatki, ki so lahko spremenljivi ali pa konstantni. Z ozirom na to delimo podatke na konstante in spremenljivke. Ime konstante ali spremenljivke pove, za katere podatke gre (temperatura, tok, napetost, ploščina...), tip spremenljivke pa določa, za kakšne podatke gre (cela števila, realna števila, znake...). Ime konstante ali spremenljivke je lahko poljubna kombinacija črk in števil, ki se začne s črko. Prevajalnik za programski jezik C loči med majhnimi in velikimi črkami. Tako sta `ime` in `Ime` dve različni spremenljivki. Konstante po dogovoru pišemo z velikimi črkami.

Vsako spremenljivko in konstanto moramo napovedati ali, kot tudi rečemo, deklarirati. Pri deklaraciji spremenljivke povemo ime spremenljivke in njen tip, ki označuje, kakšne vrednosti lahko hranimo v tej spremenljivki in katere operacije so nam na voljo.

Podatkovni tip spremenljivke pove prevajalniku koliko pomnilnika naj rezervira za posamezno spremenljivko in katere so dopustne operacije, ki jih lahko s temi spremenljivkami izvajamo. Osnovni podatkovni tipi so:

TIP	št. Bytov	velikost (mejne vrednosti)	
char	1	od -128 do 127	ASCII znak
short int	2	od -32768 do 32 767	predznačeno celo število
int	4	od -2147483648 do 2147483647 (velja za 32-bitno okolje)	predznačeno celo število
long int	4	od -2147483648 do 2147483647	predznačeno celo število
float	4	od 3.4e-38 do 3.4e38	predznačeno realno število
double	8	od 1.7e-308 do 1.7e308	predznačeno realno število
long double	10	od 3.4e-4932 do 3.4e4932	predznačeno realno število

Tabela 1: Osnovni podatkovni tipi

Primeri:

```
int x;          //spremenljivko smo napovedali (deklarirali), podpičje je obvezno
    x = 10;     //spremenljivki priredimo začetno vrednost (jo inicializiramo)
int x = 10;     //pri deklaraciji lahko spremenljivki že priredimo vrednost

int sirina, dolzina, visina; /* napoved več spremenljivk, če so le istega tipa*/

char znak = 'A'    //znak pišemo med enojna narekovaja
```

```
double realnoStevilo = 1.5;    //decimalna števila pišemo z decimalno piko
```

Prav tako moramo deklarirati konstanto. Konstanti moramo istočasno tudi prirediti vrednost.

Primer:

```
const float PI=3.14159;
```

Za imena spremenljivk ne smemo uporabljati rezerviranih besed (glej tabelo).

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

Tabela 2: Rezervirane besede

2.1 BRANJE IN PISANJE PODATKOV

Za branje in pisanje uporabljamo funkciji `printf` in `scanf`, ki se nahajata v knjižnici `stdio.h`. V prejšnjem poglavju smo s funkcijo `printf` izpisovali nize. S to funkcijo lahko izpisujemo tudi števila. Polna oblika te funkcije je namreč

```
printf("formatni niz", parameter1, parameter2, ...)
```

Če je v tem nizu znak %, se ob izpisu na tem mestu izpiše vrednost, kot jo določajo drugi parametri. Kakšna sta ta vrednost in način njenega izpisa, povejo znaki, ki sledijo procentu. To imenujemo formatno določilo.

Formatno določilo	Kaj naredi
%d	Izpiše celoštevilsko vrednost v desetiškem sistemu %5d izpiše celo št. na 5 mest natančno
%u	Zapiše nepredznačeno celo število v desetiškem sistemu
%x	Izpiše celo število v šestnajstiškem sistemu
%o	Izpiše celo število v osmiškem sistemu
%f	Izpiše realno vrednost v desetiškem sistemu
%e	Izpiše realno vrednost v eksponentni obliki
%c	Izpiše en znak
%s	Izpiše več znakov
%ld	Izpiše dolgo celo število (d določa celoštevilsko vrednost)
%lf	Izpiše dolgo realno število (f določa realno vrednost)

Tabela 3: Formatna določila

```

1 /* Izpis celih števil, crk, realnih števil */
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     int i=5;
8     char crka='A';
9     float stevilo=6.54321;
10    printf("To je celo stevilo %d, to je crka %c, %5.2f pa je realno stevilo.\n\n",i, crka, stevilo);
11    printf("%5d je celo stevilo.\n", i);
12    printf("%c je crka.\n",crka);
13    printf("%8.6f je realno stevilo.\n\n", stevilo);
14    system("PAUSE");
15    return 0;
16 }

```

a)

```

To je celo stevilo 5, to je crka A,  6.54 pa je realno stevilo.
      5 je celo stevilo.
A je crka.
6.543210 je realno stevilo.
Če želite nadaljevati, pritisnite poljubno tipko . . .

```

b)

Slika 6: Program za izpis celega števila, črke in realnega števila a) in izpis b)

Vrednost	Formatno določilo	Izpis	Vrednost	Formatno določilo	Izpis
123	%4d	123	-123	%4d	-123
123	%5d	123	-123	%5d	-123
123	%6d	123	-123	%6d	-123
123	%1d	123	-123	%1d	-123
123	%04d	0123	-123	%04d	-123
123	%05d	00123	-123	%05d	-0123
123	%06d	000123	-123	%06d	-00123
Vrednost	Formatno določilo	Izpis	Vrednost	Formatno določilo	Izpis
3.14159	%5.2f	3.14	3.14159	%4.2f	3.14
3.14159	%3.2d	3.14	3.14159	%5.1d	3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	3.14159
.1234	%4.2f	0.12	-.006	%4.2f	-0.01
-.006	%8.3f	-0.006	-.006	%8.5f	-0.00600
-.006	%.3f	-0.006	-3.14159	%.4f	-3.1416

a)

b)

Tabela 4: Izpisi celega (a) in realnega števila pri različnih formatnih določilih

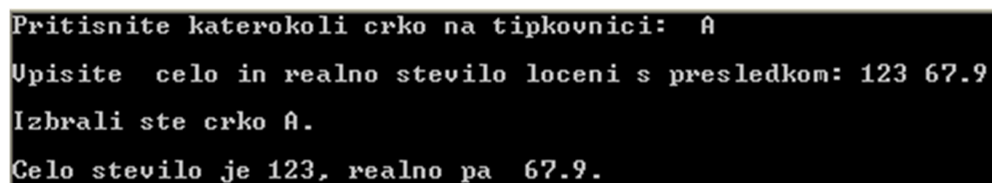
Za vnos podatkov je namenjena standardna funkcija **scanf**, ki je shranjena v knjižnici **stdio.h**. S klicem te funkcije dosežemo, da program zahteva vnos podatka, ki ga shrani v določeno spremenljivko. Funkcija **scanf** je oblike

```
scanf("formatni niz", naslov)
```

```
scanf("%tip_spremenljivke",&ime_spremenljivke);
```

Pred imenom spremenljivke se nahaja znak **&**, ki kaže na naslov spremenljivke in je obvezen.

```
1 /*Vnos podatkov s tipkovnico*/
2
3 #include <stdio.h>
4
5 int main(void)
6 {
7     char znak; int a; float b;
8     printf("Pritisnite katerokoli crko na tipkovnici: ");
9     scanf("%c", &znak);
10    printf("\nVpisite celo in realno stevilo loceni s presledkom: ");
11    scanf("%d %f", &a, &b);
12    printf("\nIzbrali ste crko %c.\n", znak);
13    printf("\nCelo stevilo je %d, realno pa %5.1f.\n\n", a, b);
14    system("PAUSE");
15    return 0;
16 }
```



```
Pritisnite katerokoli crko na tipkovnici: A
Vpisite celo in realno stevilo loceni s presledkom: 123 67.9
Izbrali ste crko A.
Celo stevilo je 123, realno pa 67.9.
```

Slika 7: Program prebere črko ter celo in realno število

S formatnim nizom povemo v kakšni obliki pričakujemo podatke. Ustrezno vrednost pa program shrani v **naslov** spremenljivke. Teh je lahko več.

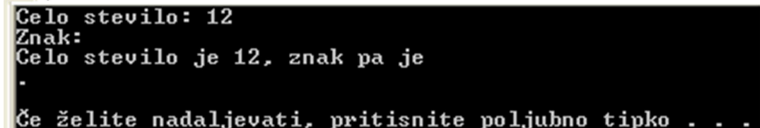
Funkcija **scanf** ne pove, da pričakuje podatek. Zato ne moremo vedeti ali program kaj računa ali čaka na nas, da bomo vnesli podatke. Da bi vedeli, ali program od nas pričakuje podatke, običajno prej najjavimo branje s funkcijo **printf**.

Pri branju znakov pa nastopijo težave. Funkcija **scanf** ne razlikuje med pritiskom na tipko **Enter** kot koncem vnosa in pritiskom na tipko **Enter** kot vsako drugo tipko. Tako znaka ne moremo prebrati, če smo prej brali kaj drugega (npr. število, slika 8).


```

1 #include <stdio.h>
2
3 int main()
4 {
5     char znak;
6     int stevilo;
7     printf("Celo stevilo: ");
8     scanf("%d", &stevilo);
9     printf("Znak: ");
10    scanf("%c", &znak);
11    printf("\nCelo stevilo je %d, znak pa je %c.\n\n", stevilo, znak);
12    system("PAUSE");
13    return 0;
14 }

```



Celo stevilo: 12
Znak:
Celo stevilo je 12, znak pa je .
.
Če želite nadaljevati, pritisnite poljubno tipko . . .

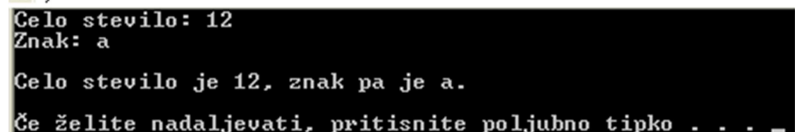
Slika 8: Branje celega števila in znaka (neuspešno) in izpis

Branje števila smo končali s pritiskom na tipko Enter. Opravili smo vnos podatka za spremenljivko `stevilo`, vendar znak, ki ustreza tipki Enter ni bil prebran. Zato ga moramo preskočiti. To storimo s formatnim določilom `%"*c"`. Napišemo `scanf ("%*"c%"c", ...)`. Na sliki 9 je popravljen primer s slike8.

```

1 #include <stdio.h>
2
3 int main()
4 {
5     char znak;
6     int stevilo;
7     printf("Celo stevilo: ");
8     scanf("%d", &stevilo);
9     printf("Znak: ");
10    scanf("%*"c%"c", &znak);
11    printf("\nCelo stevilo je %d, znak pa je %c.\n\n", stevilo, znak);
12    system("PAUSE");
13    return 0;
14 }

```



Celo stevilo: 12
Znak: a
Celo stevilo je 12, znak pa je a.
Če želite nadaljevati, pritisnite poljubno tipko . . . _

Slika 9: Branje celega števila in znaka in izpis (popravljen)

S funkcijo `scanf` lahko beremo le znake, ki sestavljajo eno besedo. Če se pojavi presledek, funkcija `scanf` konča z branjem. To pomeni, da niza sestavljenega iz več besed, ne moremo prebrati s to funkcijo. V tem primeru uporabimo funkcijo `gets()` ali funkcijo `puts()`. Vendar o tem več kasneje.

2.2 PRETVORBE MED PODATKOVNIMI TIPI

Če v izrazu nastopajo podatki različnih tipov, se pretvorijo v enak tip. Včasih to pretvorbo opravi prevajalnik, včasih pretvorbo zahteva programer. Če sta operanda pri aritmetičnih operatorjih dveh različnih tipov, se pred operacijo v splošnem operand "nižjega" tipa pretvori v "višji" tip drugega operanda. Rezultat je "višjega" tipa. Za vsak aritmetični operator veljajo naslednja pravila pretvorbe med osnovnimi podatkovnimi tipi:

- `char` in `short` se pretvorita v `int`, `float` se pretvori v `double`
- v izrazih mešanega tipa `int` in `char`, se operand tipa `char` pretvori v tip `int`
- če je en operand tipa `double`, se tudi drugi pretvori v tip `double`, če je en operand tipa `long`, se tudi drugi pretvori v tip `long`, če je en operand tipa `unsigned`, se tudi drugi pretvori v tip `unsigned`.

```
int a;  
float b = 9.81;  
a = b;      // a je celo število, zato je a = 9, odreže se decimalni del  
b = a;      // b je realno število, zato je b = 9.000000
```

```
int a;  
float b;  
a = 3.5;    // a je celo število, zato je a = 3, odreže se decimalni del  
b = 20;     // b je realno število, zato je b = 20.000000
```

Število decimalnih mest

```
char a, b, c;  
a = 'F';  
b = 'G';  
c = '+';
```

V tem primeru se v spremenljivko shrani ASCII vrednost znaka. Tako imajo spremenljivke naslednje vrednosti: `a = 70`, `b = 71` in `c = 43` (glej tabelo ASCII znakov).

Spremenljivko poljubnega tipa lahko pretvorimo v drug tip, ki ga v oklepajih navedemo pred spremenljivko. Pretvorbo opravimo z izrazom:

(ime_tipa)izraz

Vrednost izraza postane zahtevanega tipa. Tako z `(int)7.654321` izraz postane tipa `int` z vrednostjo 7.

3 OPERATORJI

3.1 PRIREDITVENI OPERATOR

Splošna oblika ukaza (sintaksa) za prirejanje vrednosti izraza spremenljivki je :

spremenljivka = izraz;

Prireditveni operator = je binaren kar pomeni da potrebuje 2 operanda:

- spremenljivko (vedno levi operand), ki ji priredimo vrednost;
- izraz (vedno desni operand je aritmetičen in/ali logičen izraz), čigar vrednost se priredi spremenljivki na levi strani.

Tako ni možno zapisati dveh spremenljivk, ne da bi med njima zapisali tudi operacijo. V tem primeru vzame prevajalnik del izraza kot novo spremenljivko, ki še ni deklarirana in javi napako.

Primer:

```
int a, b, c, d, e;  
a=b * c + de;
```

/ matematično je d*e = de, prevajalnik pa vzame de kot novi spremenljivko*/*

Izraze lahko razlikujemo po tipu operatorjev ali po tipu rezultata:

- po tipu operatorja razlikujemo aritmetične, relacijske (primerjalne) in logične izraze;
- po tipu rezultata pa aritmetične in logične izraze.

Prireditveni operator = se lahko najde tudi na desni strani ukaza za prirejanje vrednosti oz. znotraj izraza:

```
int a,b,c;
```

```
a = (b = 10) + (c = 20);    // vendar se takega zapisa raje izogibamo;
```

Lahko je tudi del logičnega izraza:

```
if ((a=20) <b)
```

Lahko pišemo tudi:

```
int a,b,c;  
a=b=c=5;    //spremenljivkam a, b in c smo priredili vrednost 5.
```

Prireditveni operator ima nižjo prioriteto kot aritmetični in logični operatorji, kar pomeni, da se najprej izvršijo vse aritmetične in logične operacije šele potem pa prirejanje.

3.2 ARITMETIČNI OPERATORJI

Aritmetični operatorji za cela in realna števila so:

OPERATOR	OPERACIJA
+	seštevanje
-	odštevanje
*	množenje
/	celoštevilsko deljenje (Desni operand mora biti različen od nič.)
%	ostanek pri deljenju (samo za cela števila)
++	povečanje
--	zmanjšanje

Če je v izrazu več operatorjev *, /, % in nobenih oklepajev, se bodo operacije izvrševale po vrsti od leve proti desni strani (% ima enako prioriteto kot * in /). Seveda sta lahko razen teh operatorjev v izrazu tudi operatorja + in - le, da takrat vrstni red izvrševanja operacij poteka glede na prioriteto operatorjev, ki je za aritmetične operatorje enaka kot v matematiki. Prioriteta operatorjev se ne spreminja z uporabo oklepajev (). Spremeni se vrstni red izvrševanja operacij.

Če delimo dve spremenljivki tipa **int** z operatorjem /, je rezultat tipa int (decimalke odreže).

$$1 / 2 = 0$$

$$5 / 2 = 2$$

$$18 / 5 = 3$$

% pa je binarni operator, ki vrne ostanek pri deljenju.

$$7 \% 2 = 1$$

$$10 \% 4 = 2$$

Zanimivo je povečanje spremenljivke za 1. V zgornjem primeru smo uporabili dva načina in sicer c++ in ++e. c++ pomeni, da kot vrednost izraza uporabimo staro vrednost c in nato c povečamo za 1. ++e pomeni, da e povečamo za 1 in novo vrednost priredimo spremenljivki y. Zato dobimo v prvem primeru c=2 in x=1, v drugem primeru pa e=2 in y=2. Enako velja za operator --, le da v tem primeru spremenljivko zmanjšujemo.

Prireditveni in aritmetični operator lahko združimo v kombiniran operator:

OPERATOR	PRIMER	EKVIVALENTEN ZAPIS
+=	a += b	a = (a + b)
-=	a -= b	a = (a - b)
*=	a *= b	a = (a * b)
/=	a /= b	a = (a / b)
%=	a %= b	a = (a % b)

```

1  /*Aritmetični operatorji*/
2
3  #include <stdio.h>
4
5  int main(void)
6  {
7      int a=4, b=8, c=1, e=1, x, y, rezultat;
8      rezultat= a + b;
9      printf("Vsota je %d. \n", rezultat);
10     rezultat= a - b;
11     printf("Razlika je %d. \n", rezultat);
12     rezultat= a * b;
13     printf("Zmnozek je %d. \n", rezultat);
14     rezultat= b / a;
15     printf("Celostevilski kvocient je %d. \n", rezultat);
16     rezultat= a % b;
17     printf("Ostane pri deljenju je %d. \n\n", rezultat);
18     printf("Ce je operator za spremenljivko, dobimo:\n");
19     x=c++;                                /*Operator za spremenljivko*/
20     printf("c =  %d    x = %d\n", c, x);
21     printf("Ce je operator pred spremenljivko, dobimo:\n");
22     y = ++e;                              /*Operator pred spremenljivko*/
23     printf("e =  %d    y = %d\n\n", e, y);
24
25     system("PAUSE");
26     return 0;
27 }

```

```

Vsota je 12.
Razlika je -4.
Zmnozek je 32.
Celostevilski kvocient je 2.
Ostane pri deljenju je 4.

Ce je operator za spremenljivko, dobimo:
c =  2    x = 1
Ce je operator pred spremenljivko, dobimo:
e =  2    y = 2

```

Slika 10: Aritmetični operatorji in izpis rezultatov

3.3 PRIMERJALNI OPERATORJI

S primerjalnimi operatorji primerjamo dva operanda. Rezultat primerjave je logična vrednost *resnično* (**true ali 1**) ali *neresnično* (**false ali 0**). Najpogosteje se z njimi srečujemo v zankah ali pogojnih stavkih.

OPERATOR	POMEN	RAZLAGA
<code>==</code>	je enako	Operator vrne vrednost resnično (true oz. 1), kadar sta oba operanda enaka.
<code>!=</code>	je različno	Pogoj je resničen, če sta operanda različna.
<code><</code>	manjši	Operator vrne resnično, če je levi operand manjši od desnega.
<code>></code>	večji	Operator vrne resnično, če je levi operand večji od desnega.
<code><=</code>	manjši ali enak	Operator vrne resnično, če je levi operand manjši ali enak desnemu.
<code>>=</code>	večji ali enak	Operator vrne resnično, če je levi operand večji ali enak desnemu.

Pri uporabi primerjalnih operatorjev moramo posebej paziti, ko preverjamo enakost. Zelo pogosta napaka je, da pozabimo en enačaj.

```
a = 5;
```

```
b = (a==10);
```

Po teh dveh izvedenih stavkih je v spremenljivki `b` vrednost 0 (ker `a` ni enak 10, je rezultat primerjave 0), v spremenljivki `a` pa 5.

Če pa se zmotimo in napišemo

```
a = 5;
```

```
b = (a=10);
```

bodo stavki še vedno sintaktično pravilni. Vendar bo vrednost spremenljivke `b` sedaj 10, prav tako bo vrednost spremenljivke `a` 10.

Relacijska operatorja `!=` in `==` imata enako prioriteto, obenem pa imata nižjo prioriteto od ostalih relacijskih. Oklepaji spremenijo vrstni red izvajanja – ne pa prioritete.

Relacijski operatorji imajo nižjo prioriteto kot aritmetični operatorji.

3.4 LOGIČNINI OPERATORJI

Logične izraze lahko kombiniramo z operatorji

OPERATOR	POMEN
<code>!</code>	logični NE
<code>&&</code>	logični IN
<code> </code>	logični ALI

Logični operator ! izvaja logično operacijo NOT. Je unarni operator, ki negira logično vrednost (true ali false) operanda. Rezultat je vedno celoštevilčen tipa int, ker je lahko samo 1 ali 0. Prioriteta logičnega operatorja ! je višja od prioritete kateregakoli drugega operatorja razen unarnega aritmetičnega operatorja -.

Pri tem veljajo enaka pravila kot pri matematiki, torej se && izvede pred ||.

4 KONTROLA POTEKA PROGRAMA

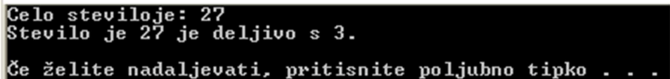
4.1 POGOJNI STAVKI (STAVKI `if`)

Pogosto postopek zahteva, da kako akcijo izvedemo le, če so izpolnjeni določeni pogoji. To nam omogoča pogojni stavek. Njegova osnovna oblika je (stavek `if`):

<code>if (pogoj)</code> <code>stavek1;</code>	<code>if (pogoj)</code> <code>{stavek1;</code> <code>stavek2;</code> <code>...</code> <code>}</code>
--	--

Če je pogoj izpolnjen (ima vrednost različno od 0), se izvede stavek1 oziroma vsi stavki med zavitima oklepajema.

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a;
6     printf("Celo stevilo je: ");
7     scanf("%d", &a);
8     if(a%3==0)
9         printf("Stevilo je %d je deljivo s 3.\n\n", a);
10
11     system("PAUSE");
12     return 0;
13 }
```



Celo stevilo je: 27
Stevilo je 27 je deljivo s 3.
Če želite nadaljevati, pritisnite poljubno tipko . . .

Slika 11: Preverjanje deljivosti števila s 3

Druga oblika pogojnega stavka (stavek `if-else`) je

<code>if (pogoj)</code> <code>stavek1;</code> <code>else</code> <code>stavek2;</code>	<code>if (pogoj)</code> <code>{stavki1}</code> <code>else</code> <code>{stavki2}</code>
--	--

Če je pogoj izpolnjen (ima vrednost različno od 0), se izvede stavek1 oziroma vsi stavki med zavitima oklepajema. Če pa pogoj ni izpolnjen (ima vrednost 0), pa se izvede stavek2 (oz. stavki2 med zavitima oklepajema).


```

1 #include <stdio.h>
2 //Preverimo ali je stevilo sodo ali liho
3 int main()
4 {
5     int a;
6     printf("Vnesi celo stevilo: ");
7     scanf("%d",&a);
8     if(a%2==0)
9         printf("Stevilo %d je sodo.\n\n",a);
10    else
11        printf("Stevilo %d je liho.\n\n",a);
12    system("PAUSE");
13    return 0;

```

```

Unesi celo stevilo: 12
Stevilo 12 je sodo.

```

```

Unesi celo stevilo: 15
Stevilo 15 je liho.

```

Slika 12: Preverjanje sodosti in lihosti števila

```

1 /*Program naj prebera vrednosti dveh uporov in na osnovi vnesene
2  vezave (zaporedna ali vzporedna) izračuna nadomestno upornost.*/
3 #include <stdio.h>
4 int main()
5 {
6     float r1, r2, rNad;
7     int vezava;
8     printf("Upornost v ohmih r1= ");
9     scanf("%f",&r1);
10    printf("Upornost v ohmih r2= ");
11    scanf("%f",&r2);
12    printf("Vnesite vezavo: 1 zaporedno, 2 vzporedno: ");
13    scanf("%d",&vezava);
14    if(vezava==1)
15        {rNad=r1+r2;
16        printf("Nadomestna upornost je %.1f ohmov.\n\n",rNad);}
17    else if(vezava==2)
18        {rNad=(r1*r2)/(r1+r2);
19        printf("Nadomestna upornost je %.2f ohmov.\n\n",rNad);}
20    else
21        printf("Neobstoječa vezava\n\n");
22    system("PAUSE");
23    return 0;
24 }

```

```

Upornost v ohmih r1= 15
Upornost v ohmih r2= 21
Vnesite vezavo: 1 zaporedno, 2 vzporedno: 1
Nadomestna upornost je 36.0 ohmov.

```

```

Upornost v ohmih r1= 15
Upornost v ohmih r2= 21
Vnesite vezavo: 1 zaporedno, 2 vzporedno: 2
Nadomestna upornost je 8.75 ohmov.

```

```

Upornost v ohmih r1= 15
Upornost v ohmih r2= 21
Vnesite vezavo: 1 zaporedno, 2 vzporedno: 5
Neobstoječa vezava

```

Slika 13: Vzporedna in zaporedna vezava dveh uporov

Else pripada prvemu stavku if nad njim. Poglejmo še sledeči primer.

```

if(n>0)
    if(a>b)
        z=a;
    else
        z=b;

```

V tem primeru spada `else` k notranjemu stavku `if(a>b)`.

```

if(n>0){
    if(a>b)
        z=a;
}
else
    z=b;

```

Če želimo, da spada `else` k zunanjemu stavku `if`, to naredimo z oklepajem.

```

if(n>0)
    if(a>b) z=a;
else
    z=b;

```

Po videzu tega zapisa bi sklepali, da spada `else` k zunanjemu stavku `if(n>0)`, vendar ni tako.

Takemu načinu pisanja se izogibamo,

4.2 STAVEK switch

Stavek `switch` je podoben več vgnezenim stavkom `if-else`. Omogoča nam, da izberemo pri katerem pogoju se bo izvršil določen stavek.

Sintaksa:

```

switch(a)          /* ni podpičja!! a je tip int ali char */
{
case 1:             /* a je enak 1 */
    operacija1;
    break;          /* izhod iz zanke */
case 2:             /* a je enak 2 */
    operacija2;
    break;          /* izhod iz zanke */
default:           /* a ni enak 1 ali 2 */
    operacija3;
    break;          /* izhod iz zanke */
}

```

Opis delovanja:

Če je vrednost operanda `a` enaka 1, se izvede *operacija1*, če je vrednost `a` enaka 2, se izvede *operacija2*, sicer se izvede *operacija3*. V primeru uporabe stavka `switch` gre za možnost uporabe več poti.

Primer:

```
switch(a)
{
case 1:          /* če je vrednost a enaka 1: */
    b=1;         /* postavimo vrednost b na 1 */
    c++;         /* povečamo c za 1 in */
    break;       /* zapustimo zanko */
case 2:          /* če je vrednost a enaka 2: */
    b=0;         /* postavimo vrednost b na 0 */
    c--;         /* zmanjšamo c za 1 in */
    break;       /* zapustimo zanko */
default:         /* če a ni enak 1 ali 2 */
    b=2;         /* postavimo vrednost b na 2 in */
    break;       /* zapustimo zanko */
}
```

```
1 #include <stdio.h>
2 /*Sešteje, odšteje, zmnoži ali zdeli dve števili. */
3
4 int main()
5 {
6     double a, b, rez;      /*števili in rezultat*/
7     char op;               /*operacija*/
8     int napaka;            /*koda napake*/
9     printf("Vnesi prvo stevilo: "); scanf("%lf", &a);
10    printf("Vnesi drugo stevilo: "); scanf("%lf", &b);
11    printf("Izberi eno izmed operacij (+, -, *, /): "); scanf("%c%c", &op);
12    printf("\n");
13
14    napaka = 0;
15    switch (op) {
16        case '+': rez = a + b; break;
17        case '-': rez = a - b; break;
18        case '*': rez = a * b; break;
19        case '/': if(b==0)napaka=1; else rez = a / b; break;
20        default: napaka=2; break;
21    }
22    if(napaka){
23        printf("Napaka: ");
24        switch(napaka){
25            case 1: printf("Deljenje z 0"); break;
26            case 2: printf("Operacije %c ne poznam", op); break;
27            default: printf("Neznana napaka"); break;
28        }
29        printf(".\n");
30    }
31    else
32        printf("Rezultat: %g %c %g = %g\n", a, op, b, rez);
33    system("PAUSE");
34    return 0;
35 }
```

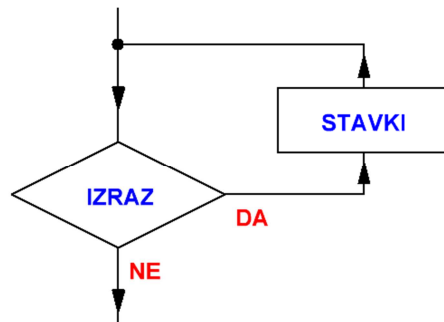
Slika 14: Računske operacije

4.3 ZANKA while

Zgradba zanke while je enostavna:

while (izraz) stavek

Dokler je logični pogoj, kot ga določa izraz, izpolnjen, izvajamo stavek. Kakor hitro pogoj ni več izpolnjen, se izvede stavek, ki sledi stavku while. Dogajanje v programu lahko prikažemo z diagramom poteka.



Slika 15: Diagram poteka za zanko while

Zako while največkrat uporabljamo kadar števila ponavljanj ne poznamo vnaprej kot v spodnjem primeru.

```
1 #include <stdio.h>
2 /*Izračunaj srednjo vrednost vpisanih števil različnih od 0.*/
3 int main()
4 {
5     int i=0;
6     float vsota=0.0, x;
7     printf(" Vpisite niz številk !=0 in niclo za konec.\n\n");
8     printf(" x[0]= "); scanf("%f",&x);
9     while (x!=0.0){
10         vsota=vsota + x;
11         printf(" x[%d]= ",++i);
12         scanf("%f",&x);
13     }
14     vsota=vsota/i;
15     printf(" Srednja vrednost = %f\n\n",vsota);
16
17     system("PAUSE");
18     return 0;
19 }
20 }
```

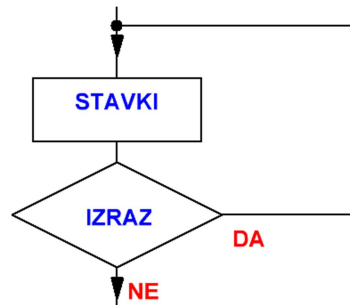
```
Upisite niz številk !=0 in niclo za konec.
x[0]= 5
x[1]= 8
x[2]= 9
x[3]= 0
Srednja vrednost = 7.333333
```

Slika 16: Računanje srednje vrednosti

4.4 ZANKA `do while`

Zanka `do while` ima obliko:

```
do  
stavek;  
while (izraz);
```



Slika 17: Diagram poteka za zanko `do while`

Stavek se bo izvrševal tako dolgo, dokler je logični pogoj, kot ga določa `izraz`, izpolnjen. Za razliko od zanke `while` se `izraz` kontrolira na koncu zanke in se zato `stavek` izvrši vsaj enkrat.

```
i=0;  
do{  
printf("%d\n", i);  
i++;  
}  
while (i<10);
```

Program izpiše številke od 0 do 9.

4.5 ZANKA `for`

Zanka `for` ima obliko:

```
for (izraz1; izraz2; izraz3 ) stavek;
```

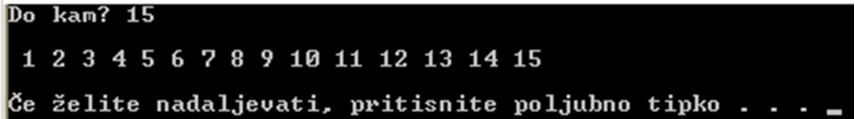
Najprej se izračuna `izraz1`. Če je pogoj, kot ga določa `izraz2`, resničen, se izvede `stavek`. Nato se izračuna `izraz3`, ponovno se preveri `izraz2`....

Z zanko `for` smo torej zapisali isto kot z zanko `while`, če le v `stavku` ne uporabimo akcije, ki spremeni normalno obnašanje zanke (`break`, `continue`, `return`).

```
izraz1;  
while (izraz2) {  
    stavek;  
    izraz3;  
}
```

Katero obliko zanke uporabimo, je bolj ali manj vseeno. Zanko `for` pogosteje uporabljamo tam, kjer štejemo ponovitve, zanko `while` pa tam, kjer načeloma ne vemo, koliko ponovitev bomo izvedli.

```
1 #include <stdio.h>
2 /*Izpiše vsa števila od 1 do n.*/
3 int main()
4 {
5     int n, stevilo;
6     printf("Do kam? ");
7     scanf("%d", &n);
8     printf("\n");
9     for (stevilo = 1; stevilo <= n; stevilo++)
10         printf(" %d", stevilo);
11     printf("\n\n");
12     system ("PAUSE");
13     return 0;
14 }
15
```



```
Do kam? 15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Če želite nadaljevati, pritisnite poljubno tipko . . . _
```

Slika 18: Izpisovanje števil od 1 do n