Uvod v programski jezik C

Za začetek bo opisan način pisanja programov v programskem jeziku C. Pri samem pisanju lahko vedno uporabljamo ASCII znake (črke, številke, ločila, ...), v nekaterih primerih pa tudi grafične znake, ki pa lahko nastopajo le v izpisih. Znakov *š*, *č*, *ž* ipd. ne uporabljamo za imena spremenljivk, funkcij ali ukazov. C loči med velikimi in malimi črkami!

Nekateri znaki imajo točno določen pomen:

Znak	Pomen
/*	Začetek komentarja (komentar se mora končati z zaporedjem */)
*/	Konec komentarja
#	Makro ukazi (deklaracije konstant, ukazi prevajalniku, ...)
;	Zaključek programske vrstice
,	Naštevanje
\	Posebni znaki za krmiljenje izpisa
0x	Sledi izpis šestnajstiškega števila
"	Začetek in konec spremenljivke tipa *string* (več zaporednih znakov)
'	Začetek in konec spremenljivke tipa *char* (znak)

Primer:

/ kar je napisano tukaj, ne vpliva na potek programa */*

1 Tipi podatkov

Prvi znak v imenu spremenljivke ali konstante ne sme biti številka, lahko je samo črka ali znak za podčrtaj (_). Sicer lahko ime spremenljivk, konstant ipd. sestavljajo črke (ASCII), številke in znak _.

1.1 Preprosti tipi podatkov

Tip podatkov	*Velikost v bitih*	*Obseg*
char	8	-128 do 127
signed char	8	-128 do 127
unsigned char	8	0 do 255
short int	16	-32768 do 32767
unsigned int	16	0 do 65535
int	16	-32768 do 32767
long	32	-2147483648 do 21474836487
unsigned long	32	0 do 4294967295
float	32	3.4×10^{-38} do 3.4×10^{38}
double	64	1.7×10^{-308} do 1.7×10^{308}
long double	80	3.4×10^{-4932} do 1.1×10^{4932}

1.2 Spremenljivke

Spremenljivke v programskem jeziku C lahko zajemajo podatke zgoraj opisanih tipov. Da jih lahko uporabimo, jih je potrebno najprej deklarirati, kar storimo tako, da najprej navedemo tip, potem pa še ime spremenljivke:

tip_spremenljivke ime_spremenljivke;

Glede na doseg delimo spremenljivke na lokalne in globalne. Lokalne spremenljivke veljajo samo znotraj funkcij, v katerih so definirane, globalne pa v celotnem programu. Globalne spremenljivke večinoma dobimo tako, da jih definiramo zunaj funkcij.

Primeri:

```
int i;                  /* deklaracija celoštevilčne spremenljivke i */
float f=0.2;            /* deklaracija in inicializacija spremenljivke f */
double d1, d2, d3=-2.3; /* deklaracija spremenljivk d1, d2 in d3 ter inicializacija spremenljivke d3 */
```

1.3 Konstante

Konstanta je vrednost, ki je med delovanjem programa na splošno ni mogoče spreminjati.

Konstanta	*Format*	*Primeri*
Znak	V enojnih narekovajih	'a','!'
Celo število	Desetiški (Decimalni): Niz znakov (števil), prvi znak ni nič Osmiški (Oktalni): Niz znakov (števil), prvi znak je 0, sledijo osmiške številke Šestnajstiški (Heksadecimalni): Niz znakov, prva znaka sta 0x, sledijo osmiške številke	12, 69 077, 035 0xAB, 0x3C
Število s plavajočo vejico	Desetiški (Decimalni): [število].[število] Eksponencialni: [število].[število][E|e][+|-][število]	1.3, 1.6 4.2e-7, 0.1E8
String	Zaporedje znakov v dvojnih narekovajih: "znak,znak,..."	"Dober dan", "C\n"

Možnosti za definicijo konstante so različne, prva je zapis z makro ukazi:

```
#define SPR 12    /* zaporedje znakov SPR je v procesu prevajanja nadomeščeno z 12 */
```

Druga možnost je deklaracija:

```
const char xx='1';          /* konstanti tipa char je prirejen znak '1' */
```

1.4 Vektorji in polja

Kot vsak programski jezik tudi C omogoča operacije z vektorji in polji. Vektorju oz. polju je potrebno med deklaracijo prirediti ime in število elementov.

Število elementov vektorja oz. polja je omejeno z velikostjo pomnilnika.

Primeri za vektor:

```
float a[17];                /* vektor tipa float s 17 elementi */
int b[3]={1,2,3};           /* primer prireditve vrednosti vektorju */
b[0]=1;b[1]=2;b[2]=3;       /* prireditev vrednosti za elemente vektorja */
```

Polje sestavlja več enakih vektorjev. Na tak način ga tudi deklariramo in mu prirejamo vrednosti.

Primeri za polje:

```
float a[2][4];                  /* polje tipa float z 2x4 elementi */
int b[2][3]={{1,2,3},{4,5,6}};  /* primer prireditve vrednosti polju */
b[0][0]=1;b[0][1]=2;b[0][2]=3;b[1][0]=4;b[1][1]=5;b[1][2]=6;  /* prireditev vrednosti za elemente polja */
```

1.5 Strukture

Strukture so v osnovi zelo podobne vektorjem, vendar jih za razliko od le-teh sestavljajo elementi različnih tipov.

Primer strukture:

```
struct Osebni_podatki           /* deklaracija strukture */
{
        char ime[20];           /* ime ima lahko največ 20 znakov */
        char priimek[20];       /* priimek ima lahko največ 20 znakov */
        int leto_rojstva;       /* letnica rojstva */
        unsigned spol:1;  /* spol - 0:moški, 1:ženski - spremenljivka zasede le en bit */
};
struct Osebni_podatki osebni_podatki;  /* definicija strukture */
osebni_podatki.leto_rojstva=1900;   /* prireditev vrednosti elementu strukture */
```

1.6 Kazalci

Ena od značilnosti programskega jezika C so kazalci. Kazalec je spremenljivka, ki vsebuje naslov neke druge spremenljivke ali funkcije. Deklaracija kazalca določa, na kateri tip podatkov kaže:

tip_na_katerega_kaže *ime_kazalca;

Da dosežemo objekt, na katerega kaže kazalec, uporabimo ime kazalca, pred katero postavimo znak zvezdica (*). Po drugi strani je mogoče naslov spremenljivke dobiti tako, da pred ime spremenljivke postavimo znak &.

Primeri:

```
int *int_kaz;               /* kazalec na celo število */
float *float_kaz;  /* kazalec na število s plavajočo vejico */
*float_kaz=3.01; /* vrednost spremenljivke, ki jo kaže kazalec float_kaz je 3.01*/
float_kaz=&float_spr;       /* kazalec kaže na spremenljivko float_spr */
*float_kaz=float_spr;       /* spremenljivka, ki jo kaže kazalec float_kaz se postavi na vrednost spremenljivke float_spr */
```

Iz primerov lahko razberemo, so možnosti uporabe kazalcev različne.

1.7 Razredi spremenljivk *static*, *register* in *volatile*

Včasih postavljamo za spremenljivke posebne zahteve. V takih primerih pogosto uporabimo razrede spremenljivk *static*, *register* in *volatile*. Deklaracija spremenljivk je v takih primerih zapisana na naslednji način:

static tip_spremenljivke ime_spremenljivke;
register tip_spremenljivke ime_spremenljivke;
volatile tip_spremenljivke ime_spremenljivke;

Če uporabimo razred *static*, to pomeni, da se vrednost spremenljivke ohrani tudi po izhodu iz funkcije.

Razred *register* pove prevajalniku, da gre za pogosto uporabljeno spremenljivko, ki jo je smiselno spraviti v register procesorja, s čemer se poveča hitrost programa.

Razred *volatile* pa prevajalniku pove, da se lahko spremenljivka spreminja v procesih, ki tečejo v ozadju in je torej ni dovoljeno optimizirati.

2 Aritmetični in logični operatorji

2.1 Splošni operatorji

Operator	Pomen	Primer
=	prirejanje - priredi spremenljivki na levi strani vrednost izraza na desni	a=b
[]	izbor elementa v polju	a[4]
.	izbor elementa strukture	a.b
->	izbor elementa strukture, na katero kaže kazalec	a->b
sizeof	velikost objekta	sizeof(a)
sizeof	velikost tipa	sizeof(int)
?:	pogojni operator	(a?b:c)
,	vejica	a,b
*	vrednost naslova	*a
&	naslov	&a

2.2 Aritmetični operatorji

Operator	Pomen	Primer
+	seštevanje	a+b
-	odštevanje	a-b
*	množenje	a*b
/	deljenje	a/b
%	modulo (ostanek celoštevilčnega deljenja)	a%b
++	pred inkrement (poveča vrednost spremenljivke za 1 in jo nato uporabi)	++a
++	po inkrement (uporabi spremenljivko in ji nato poveča vrednost za 1)	a++
--	pred dekrement (zmanjša vrednost spremenljivke za 1 in jo nato uporabi)	--a
--	po dekrement (uporabi spremenljivko in ji nato zmanjša vrednost za 1)	a--
-	unarni minus	-a
+	unarni plus	+a
+=	seštevanje in prirejanje (a+=b pomeni a=a+b)	a+=b
-=	odštevanje in prirejanje (a-=b pomeni a=a-b)	a-=b
*=	množenje in prirejanje (a*=b pomeni a=a*b)	a*=b
/=	deljenje in prirejanje (a/=b pomeni a=a/b)	a/=b
%=	modulo in prirejanje (a%=b pomeni a=a%b)	a%=b

2.3 Logični operatorji

Operator	Pomen	Primer
&	bitni IN (AND)	a&b
|	bitni ALI (OR)	a|b
^	bitni ekskluzivni ALI (XOR)	a^b
~	komplement	~a
!	negacija	!a
<<	pomik levo (a<<b pomakne a za b bitov v levo)	a<<b

>>	pomik desno (a>>b pomakne a za b bitov v desno)	a>>b
<	manjše kot	a<b
>	večje kot	a>b
==	enako	a==b
!=	različno	a!=b
>=	večje ali enako kot	a>=b
<=	manjše ali enako kot	a<=b
&&	logični IN (AND)	a&&b
||	logični ALI (OR)	a||b
&=	bitni IN (AND) in prirejanje	a&=b
|=	bitni ALI (OR) in prirejanje	a|=b
^=	bitni ekskluzivni ALI (XOR) in prirejanje	a^=b
<<=	pomik levo (a<<b pomakne a za b bitov v levo) in prirejanje	a<<=b
>>=	pomik desno (a>>b pomakne a za b bitov v desno) in prirejanje	a>>=b

2.4 Oklepaji

V programskem jeziku C imajo oklepaji več funkcij. Poleg funkcije, ki jo uporabljajo v matematiki, nastopajo še pri argumentih funkcij in eksplicitnem pretvarjanju tipov.

3 Kontrola poteka programa

Potek programa v praksi nikoli ni enosmeren, temveč so v njem vedno zajete odločitve, ker želimo različno delovanje v odvisnosti od podanih parametrov. Pogosto je tudi, da želimo nek del programa izvesti večkrat.

Takšna in podobna opravila nam omogočajo v nadaljevanju opisane kontrolne strukture.

3.1 Stavek *if-else*

Sintaksa:

```
if(pogoj)                    /* ni podpičja!!! */
{
        operacija1;          /* DA */
}
        else                 /* ni podpičja!!! */
{
        operacija2;          /* NE */
}
```

Opis delovanja:

Če je *pogoj* izpolnjen, se izvede *operacija1*, sicer se izvede *operacija2*.

Primera:

```
/* Če je a>b, se poveča za 1, sicer se postavi na 0 */
if (a<b)
{
        a=a+1;
}
else
```

```
{
        a=0;
}
```

if(a==b) a=1; / Če je a enak b* ***(pazi: operator enakosti je drugačen kot operator prireditve!!****), postavi a na 1*/*

3.2 Stavek *switch*

Sintaksa:

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

3.3 Zanka *for*

Sintaksa:

```
for(inicializacija;test;korak)  /* ni podpičja! */
{
        operacija1;
}
```

Opis delovanja:

Na začetku se izvede operacija *inicializacija*, nato pa se operacija *operacija1* izvaja tako dolgo, dokler je operacija *test* pravilna. V vsakem koraku se izvede še operacija *korak.*

Funkcija *for* je v programskem jeziku C bistveno močnejša kot v drugih jezikih!

Primera:

```
for(i=0;i<40;i++)       /* Na začetku postavimo spremenljivko i na 0 in jo večamo za 1 tako dolgo, dokler
je manjša kot 40.*/
{
        a[i]=i;         /* element vektorja a se postavi na i */
}

for(i=0;i<40;i++) b++;  /* 10 krat povečamo b za 1 */
```

3.4 Zanka *while*

Sintaksa:

```
while(pogoj)                /* ni podpičja! */
{
        operacija1;
}
```

Opis delovanja:

Operacija *operacija1* se izvaja tako dolgo, dokler je izpolnjen pogoj *pogoj*. Najprej se izvede preverjanje pogoja, šele nato se izvrši operacija.

Primeri:

```
while(i<10)                 /* dokler je i<10, se povečujeta i in j */
{
        i*=2;               /* i=i*2 */
        j+=i;               /* j=j+i */
}

while(i++<10);              /* zanka se izvaja tako dolgo, dokler je i<10, gre samo za zakasnitev */

while(1) i++;               /* neskončna zanka, v vsakem koraku se i poveča za 1 */
```

3.5 Zanka *do-while*

Sintaksa:

```
do                          /* ni podpičja! */
{
        operacija1;
} while(pogoj);
```

Opis delovanja:

Operacija *operacija1* se izvaja tako dolgo, dokler je izpolnjen pogoj *pogoj*. Najprej se izvrši operacija, šele nato se izvede preverjanje pogoja. *Operacija1* se vedno izvede vsaj enkrat!

Primer:

```
do
{
        i*=2;                   /* i=i*2 */
        j+=i;                   /* j=j+i */
}while(i<10)                    /* dokler je i<10, se povečujeta i in j */
```

3.6 Stavka *break* in *continue*

Sintaksa:

```
break;
continue;
```

Opis delovanja:

Stavka *break* in *continue* uporabljamo za prekinjanje in nadaljevanje zanke. Poleg stavka *switch* lahko s stavkom *break* prekinemo tudi katero koli drugo funkcijo.

S stavkom *continue* se pomaknemo na najbolj notranji oklepaj (}), in od tam se izvajanje zanke nadaljuje.

Primer:

```
for(i=0:i<20;i++)
{
        j=i+k;
        if(j==4) break;         /* če je j enako 4, je zanka prekinjena, program se nadaljuje po njej */
}

for(i=0:i<20;i++)
{
        j=i+k;
        if(j==4) continue;      /* če je j enako 4, se program se nadaljuje od najbolj notranjega zaklepaja
                                   (}) naprej */
}
```

4 Funkcije

Zaradi večje preglednosti in modularnosti razdelimo program v manjše dele, v primeru programskega jezika C te gradnike imenujemo funkcije. Funkcije lahko imajo poljubno število vhodov, vendar samo en izhod! Ta problem rešimo z uporabo kazalcev.

4.1 Sintaksa

```
tip_funkcije ime_funkcije(tip_vhoda1 vhod1_fn, tip_vhoda2 vhod2_fn, ...)        /* ni podpičja! */
{
tip_spr1 spr1;
tip_spr2_spr2;
        stavek1;
        stavek2;
        ⋮
```

```
        return izhod_funkcije;
}
```

Tip funkcije *tip_funkcije* ustreza tipu izhoda funkcije (*izhod_fn*). Ime funkcije lahko vsebuje enake znake kot imena spremenljivk (črke, številke in *)*. Ko izvedemo klic funkcije, se izvršijo stavki iz telesa funkcije (*stavek1*, *stavek2*, ...), na koncu pa ob izhodu iz funkcije le-ta vrne vrednost, ki jo nastavimo z ukazom *return*, to je *izhod_funkcije*.

Seveda obstajajo tudi funkcije, ki nimajo vhodov in funkcije, ki ne vrnejo izhodne vrednosti. V takem primeru uporabljamo rezervirano besedo *void*, ki nadomesti tip podatkov izhoda oz. vhode.

Znotraj telesa funkcije definiramo notranje spremenljivke, ki so definirane samo v funkciji (*spr1*, *spr2*, ...). To so t.i. *lokalne spremenljivke*.

4.2 Deklaracija funkcij

V primeru deklaracije funkcije dejansko ponovimo prvo vrstico zapisa funkcije, vendar jo moramo zaključiti s podpičjem:

tip_funkcije ime_funkcije(tip_vhoda1 vhod1_fn, tip_vhoda2 vhod2_fn, ...);

Tukaj ni potrebno navajati imen vhodov (*vhod1_fn*, *vhod2_fn*), saj se ta spreminjajo glede na potrebe programerja, pomemben je le tip, ki ga je potrebno navesti zaradi rezervacije prostora.

4.3 Klic funkcij

Ko izvajamo klic funkcije, uporabljamo naslednjo sintakso:

rezultat_funkcije=ime_funkcije(vhod1_fn, vhod2_fn, ...);

Paziti moramo, da tipi vhodov in izhoda ustrezajo tistim iz definicije, saj lahko v nasprotnem primeru pride do neprijetnih napak, ki jih je včasih zelo težko odkriti.

4.4 Funkcija main

Funkcija *main* ima med vsemi funkcijami v programu posebno mesto, saj predstavlja glavno funkcijo v programu, torej funkcijo, ki posredno ali neposredno kliče vse ostale. Tudi funkcija *main* pogosto vrne vrednost, kar pa ni nujno.

Ko poženemo program torej dejansko poženemo funkcijo *main*, ki potem po potrebi poganja ostale funkcije.

4.5 Primeri

Deklaracija funkcije:

```
float vsota(float s1, float s2);
```

Klic funkcije:

```
j=vsota(a,b);                /* klic funkcije, ki izračuna vsoto dveh spremenljivk , s1=a, s2=b */
```

Funkcija:

```
float vsota(float s1, float s2)        /* funkcija, ki izračuna vsoto dveh števil s plavajočo vejico */
{
        return (s1+s2);
}
```

Deklaracija funkcije:

```
int funkcija(int *a);
```

Klic funkcije:

```
j=funkcija(&i);              /* klic funkcije, kjer za vhod uporabimo kazalec */
```

Funkcija:

```
int funkcija(int *a)         /* vhod funkcije je kazalec tipa int, funkcija vrne vsebino kazalca */
{
        a+=2;                /* vrednost spremenljivke na katero kaže kazalec, se poveča za 2 */
```

```
        return *a;
}
```

Deklaracija funkcije:

```
void zamenjaj(int s1, int s2);      /* funkcija nima izhoda, je tipa void */
```

Klic funkcije:

```
zamenjaj(&a,&b);                    /* klic funkcije, ki zamenja vrednosti dveh spremenljivk, izhoda ni */
                                    /* vhoda funkcije sta naslova spremenljivk, ki ju želimo zamenjati */
```

Funkcija:

```
void zamenjaj(int *s1, int *s2)             /* delamo s kazalci, saj lahko funkcija vrne le eno spremenljivko */
{
int zacas;                          /* zacasna spremenljivka */
        zacas=*s1;                  /* delamo s kazalci - zamenjamo vrednosti na naslovih spremenljivk */
        *s1=*s2;
        *s2=zacas;
}                                   /* ukaza return ne potrebujemo, ker je funkcija tipa void in ne vrne ničesar */
```

5 Makro ukazi

Makro ukazi so ukazi, ki jih da programer prevajalniku. Vplivajo predvsem na potek prevajanja in le posredno na potek programa.

5.1 Ukaz #define

Sintaksa:

#define NIZ_ZANKOV niz_znakov

Makro ukaz *#define* priredi nekemu nizu znakov *NIZ_ZNAKOV* drugi niz znakov *niz_znakov*. V postopku prevajanja prevajalnik nadomesti *NIZ_ZNAKOV* z *niz_znakov*. *niz_znakov* je lahko tudi konstanta.

Primeri:

```
#define PI 3.14     /* prireditev števila 3.14 nizu znakov PI */
#define end }               /* prireditev znaka } nizu znakov end */
#define begin {             /* prireditev znaka { nizu znakov begin */
```

5.2 Ukaz *#include*

Ukaz *#include* uporabljamo za vključevanje t.i. *headerjev* v program. *Header* je datoteka (ponavadi je tipa .h), v kateri so definirane funkcije, ki jih uporabljamo v programu. V nekaterih primerih gre za datoteke, ki jih napišemo sami, v drugih pa uporabimo že napisane datoteke, ki jih dobimo od proizvajalca prevajalnika.

```
#include "header.h"         /* vključi glavo (header) header.h iz lokalnega direktorija */
#include <header.h>         /* vključi glavo (header) header.h, ki jo podaja proizvajalec programske opreme */
```

Najpogosteje uporabljamo datoteke:

stdio.h	standardni vhodi in izhodi - izpisi in zajemanje podatkov
conio.h	vhodno/izhodne rutine konzole (tipkovnice)
math.h	matematične funkcije

string.h delo s stringi

5.3 Ukazi #if, #ifdef, #else in #endif

Sintaksa:

```
#if T_ALI_F
      blok_programa1
#else
      blok_programa2
#endif

#ifdef IME1
      blok_programa3
#else
      blok_programa4
#endif
```

Ukazi vplivajo na potek prevajanja. Če je nizu znakov *T_ALI_F* prirejena vrednost večja od 0, se prevede *blok_programa1*, sicer pa *blok_programa2*. Če je definiran niz znakov IME1 (mu je prirejen kateri koli niz znakov, vključno s praznim nizom), se prevede *blok_programa3*, sicer pa *blok_programa4*.

6 Funkciji printf in scanf

6.1 Funkcija *printf*

Sintaksa:

```
#include <stdio.h>
int printf(const char *format[, argument, ...]);
```

Opis:

printf(control_f, arg1, arg2, ...)

Funkcija na standardni izhod (stdout, ponavadi zaslon) izpiše formatiran izpis v obliki, kakršno zahteva *control_f*. V primeru uspešne izvedbe vrne število izpisanih bytov, ob napaki pa EOF.

V kontrolnem nizu imajo formatna določila formata naslednjo obliko:

% [flags] [širina] [.prec] [h|l|L] tip_char

Vsako formatno določilo se začne z znakom %, po tem znaku pa v zapisanem zaporedju sledijo še ostali:

Komponenta	Zahtevan	Pomen
[flags]	Ne	Znak - (minus), ki določa levo poravnavo polja izpisa
[širina]	Ne	Širina izpisa - minimalno število znakov, ki naj jih funkcija izpiše. Funkcija lahko izpiše več znakov, če je znakov manj, jih poravna glede na argument [flags].
[.prec]	Ne	Natančnost - maksimalno število znakov, ki naj jih funkcija izpiše oz. pri argumentih tipa *float* ali *double* število izpisanih decimalnih mest.
[h|l|L]	Ne	Oznaka velikosti : *l,L* - argument je tipa *long*, *h* - argument je tipa *short*
tip_char	Da	Znak za tip: d - Argument (celo število) se izpiše v decimalni obliki. o - Argument (celo število) se izpiše v osmiški obliki. x - Argument (celo število) se izpiše v šestnajstiški obliki.

		u - Argument (nepredznačeno celo število) se izpiše v desetiški obliki brez predznaka. c - Izpiše se en znak. s - Izpiše se niz znakov (*string*). Znaki se izpisujejo, dokler funkcija ne naleti na znak \0 ali dokler se ne izpiše poodano število znakov. f - Argument (število s plavajočo vejico) se izpiše v obliki [-]mmm.nnnn, kjer je število n-jev določeno s številom decimalnih mest. V formatnem določilu podana natančnost ne vpliva na število veljavnih številk realnega števila. e - Argument (število s plavajočo vejico) se izpiše v obliki [-]m.nnnn[±]xx, kjer je število n-jev določeno s številom decimalnih mest. g - Argument (število s plavajočo vejico) se izpiše v obliki %f ali %e v odvisnosti od tega, kateri izpis je krajši.

Primer:

float a=1.;

int b=3;

printf("\na=%f, b=%d",a,b); / gre v naslednjo vrstico, izpiše: a=1.0 b=3*/*

6.2 Funkcija *scanf*

Sintaksa:

#include <stdio.h>

int scanf(const char *format[, address, ...]);

Opis:

scanf(control_f, arg1, arg2, ...)

Funkcija bere znake z vhoda in jih interpretira v skladu s formatnimi določili v nizu *control_f*. Rezultat shrani v ustrezne argumente. Kontrolni niz bo opisan v nadaljevanju, ostali argumenti pa morajo biti kazalci na objekte, v katere naj se shrani rezultat pretvorbe.

V kontrolnem nizu imajo formatna določila formata naslednjo obliko:

% [*] [širina] [F|N] [h|l|L] tip_char

Vsako formatno določilo se začne z znakom %, po tem znaku pa v zapisanem zaporedju sledijo še ostali:

Komponenta	**Zahtevan**	**Pomen**
[*]	Ne	Funkcija preskoči zaporedje znakov ustreznega argumenta.
[širina]	Ne	Širina vpisa - največje število znakov, ki naj jih funkcija prebere. Funkcija lahko prebere manj znakov, če je v nizu presledek ali znak, ki ga ni mogoče pretvoriti.
tip_char	Da	Znak za tip: d - funkcija pričakuje celo število v decimalni obliki, argument mora biti kazalec na celo število. o - funkcija pričakuje celo število v osmiški obliki, argument mora biti kazalec na celo število x - funkcija pričakuje celo število v šestnajstiški obliki, argument mora biti kazalec na celo število h - funkcija pričakuje celo število tipa short obliki, argument mora biti kazalec na celo število c - funkcija pričakuje en znak, argument mora biti kazalec na znak. V tem primeru

		funkcija ne preskoči presledkov! s- funkcija pričakuje string (niz znakov), argument mora biti kazalec na vektor znakov. Na konec niza funkcija samodejno doda znak \0. f - funkcija pričakuje število s plavajočo vejico, argument mora biti kazalec na število s plavajočo vejico. e - funkcija pričakuje število s plavajočo vejico v eksponentni obliki, argument mora biti kazalec na število s plavajočo vejico. g - funkcija pričakuje število s plavajočo vejico, argument mora biti kazalec na število s plavajočo vejico.

Primer:

```
int i;
scanf("%d",&i);  /* prebere vrednost spremenljivke i s tipkovnice */
```

7 Zaključek

V gradivu so podane le osnovne funkcije in lastnosti programskega jezika C, ki jih potrebujemo za začetek dela. Podrobnejši opis je mogoče najti v podani literaturi.

8 Literatura

[1] Matjaž Prtenjak, C++ za velike in male, 1. izd., Izola : Desk, 1995, ISBN 961-6002-31-7

[2] B.W. Kernighan, D.M. Ritchie, Programski jezik "C", 4. izd., Ljubljana: Fakulteta za elektrotehniko, 1994, prevajalec: L. Mlakar, ISBN 86-7739-053-7

[3] B. Babet, Borland C++ : vodič za profesionalne programere, Zagreb: Znak, 1995, prevajalec: D. Kukoleča, ISBN 953-6185-38-5

[4] C and C++ tutorials, http://www.gustavo.net/programming/c__tutorials.shtml

[5] C Programming, http://www.strath.ac.uk/CC/Courses/NewCcourse/ccourse.html

[6] Programming in C, http://www.cs.cf.ac.uk/Dave/C/CE.html

[7] Brian W. Kernighan: Programming in C: A Tutorial, http://www.csc.lsu.edu/tutorial/ten-commandments/bwk-tutor.html

[8] Howstuffworks.com's "How C Programming Works", http://www.howstuffworks.com/c.htm

[9] C Programming Contents, http://gd.tuwien.ac.at/languages/c/programming-bbrown/

[10] An Introduction to C Programming, http://www.cit.ac.nz/smac/cprogram/

9 Dodatek

Tabela ASCII znakov:

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f