



C/C++

---

Andrej Bagon  
TŠC Nova Gorica



# Kaj je to C?

---

- Programski jezik, katerega je izdelal leta 1972 Dennis Ritchie v AT&T Bell Laboratorijih
- Poimenoval ga je C, ker je obstajal že takrat programski jezik A in B, torej naslednji programski jezik, je dobil ime C.
- C je višji programski jezik in tudi eden od najbolj uprabljenih višjih programskih jezikov

# Lastnosti programskega jezika



## C

---

- Enostaven za branje
- Vzdrževanje programov je enostavno
- Prenos programov iz ene platforme (operacijskega sistema) na drugo platformo je enostavno.
- Da program prevedemo v strojni jezik potrebujemo prevajalnik.



# Programski jezik C

---

- Veliko drugih programskih jezikov je napisanih v programskem jeziku C
- Primeri: c, c++, perl, python, php,...
- Da se naučimo programskega jezika C ne potrebujemo predznanja iz drugih programskih jezikov, je pa vsako predhodno znanje takega drugega programskega jezika samo lažje delo da se naučimo programskega jezika C



# Izdelava programov

---

- Da izdelate program narejen v programskem jeziku C potrebujete prevajalnik za C
- MS Visual C, Borland C Builder, GNU C, Intel C,...
- Računalnik: Procesor 386, 8mb ram, 500mb disk z linux operacijskim sistemom



# Prvi C program

---

- #include direktiva
- Glave programa (header files)
- Komentari
- main() funkcija
- return stavek
- exit() funkcija
- Nova vrstica '\n'
- Podatkovni tip void
- Prevajanje programa



# Zdravo C program

---

```
/* prvi.c: To je moj prvi C program */  
#include <stdio.h>  
int main(void)  
{  
    printf("Zdravo! To je prvi program v Cju!\n");  
    return 0;  
}
```



# Prvi program v Cju

---

- Shranimo ga pod prvi.c – končnica mora biti .c
- V unix sistemih ta program prevedemo z ukazom `cc prvi.c -o prvi` in prevajalnik nam izdela datoteko, ki jo lahko poženemo z ukazom `./prvi`
- V windows sistemih kliknemo ikono prevajaj in programsko okolje nam prevede program in zažene.





# Komentarji 1

---

- Prva vrstica v programu je bil komentar:  
`/* prvi.c: To je moj prvi C program */`
- Komentarji se začnejo z `/*` in končajo z `*/`. Vse kar je vmes se ne prevaja – se ignorira.
- Zakaj vključujemo komentarje v program – za lažje razumevanje, ko program drugač beremo.



## Komentarji 2

Večina C prevajalnikov vam omogoča komentarje čez več vrstic

/\* to je prva vrstica komentarja

to je druga vrstica komentarja

to je tretja vrstica komentarja \*/

kar je enako

/\* to je prva vrstica komentarja \*/

/\* to je druga vrstica komentarja \*/

/\* to je tretja vrstica komentarja \*/



# Komentarji 3

---

- Nekateri C prevajalniki omogočajo uporabo komentarjev iz C++ - PAZI NEKATERI NE VSI!

// to je prvi komentar

// to je drugi komentar

// to je tretji komentar



# Komentarji 4

---

- Vgnjezdeni komentarji niso dovoljeni v ANSI C standardu.

/\* to je prvi komentar

/\* to je drugi komentar \*/

to je tretji komentar \*/

Prikazani stavki niso dovoljeni v programskem jeziku C

- Komentarje lahko uporabimo tudi pri testiranju našega programa – komentiramo dele, ki ne delajo



# Direktiva `#include`

---

- Druga vrstica  
`#include <stdio.h>`
- V Cju pomeni `#include` navodilo preprocesorju da pogleda za datoteko, ki jo navedemo za direktivo (v našem primeru datoteka `stdio.h`)
- Preprocesor je del prevajalnika, ki pripravi program preden se prevaja.



# Direktiva `#include`

---

- Za `#include` sledi `<stdio.h>`
- Torej vključujemo datoteko `stdio.h`
- Stdio – pomeni standard input-output – datoteka vsebuje prototipe in makroje za delo z IO.



# Pazi!

---

- Programski jezik je občutljiv na velike in majhne črke. Torej če napišemo `MAIN()` ali `Main()` ali `main()` je čisto nekaj drugega (različno kot pri pascalu, kjer ni važno kako napišemo)!



# Glave programa - header

---

- Datoteke kot `stdio.h` so datoteke, ki jih imenujemo headerji. To so datoteke, ki jih vedno vključimo na vrhu c programa. Končnica `h` tudi pomeni header.
- Poleg headerja `stdio.h` poznamo tudi druge, kot so `stdlib.h`, `string.h`, `math.h`,...





# #include in <> ali ""

---

- V #include direktivo lahko vnesemo datoteko v <> oklepajih, kot tudi v "" oklepajih

Razlike:

- <> pomeni, da gledamo zunaj trenutnega direktorija/ponavadi v direktoriju, kjer imamo prevajnik
- "" pomeni, da gledamo na trenutnem direktoriju
- Navadno so headerji shranjeni v direktoriju include/



# main() funkcija 1

---

- Specialna funkcija v Cju. Pomeni začetek programa (glavni blok programa).
- Funkcijo main() lahko postavimo bilokam v program, ampak vedno se funkcija main() prva začne z izvajanjem (torej ponavadi je na koncu programa)
- Konec programa je, ko se funkcija main() konča.
- Torej začetek, kot tudi konec programa je v funkciji main().



## main() funkcija 2

---

- Vsak blok v programskem jeziku C se začne z { in konča z } – enakovredno ukazom begin in end v pascalu.
- Znotraj našega programa kličemo še funkcijo printf, ki je del stdio modula.



# Nova vrstica - printf

---

- V programskem jeziku C nimamo možnosti uporabe `write` in `writeln` (razlika je skok v naslednjo vrstico), ampak moramo sami izvesti skok v naslednjo vrstico.
- To nam omogoča, da v tekst vnesemo znak `'\n'`, kar pomeni, da na tem mestu se program postavi v naslednjo vrstico.



# return

---

- Vse funkcije v Cju navadno vračajo vrednosti.
- Funkcije vračajo vrednost s pomočjo funkcije return
- `return 0;` pomeni, da se je program pravilno končal.
- Tip vrednosti, ki se vrne ob uporabi return funkcije se napove pri deklaraciji funkcije



# void podatkovni tip

---

- Programski jezik C pozna tudi podatkovni tip void. To je podatkovni tip, ki pomeni nič!
- Kaj to dejansko pomeni – to pomeni (za naš primer), da funkcija main nima vhodnih podatkov.
- Če zapišemo void ime\_funkcije(void) pomeni, da program ne sprejema nobenega parametra in nobenega parametra ne vrača. Podobno proceduram v pascalu, kjer nič ne vračajo



# Prevajanje programov

---

- Linux (na atena)

```
# cc prvi.c -o prvi
```

- Program zaženemo

```
# ./prvi
```

- Widnows

Pritisnemo na compile in run in program se prevede in zažene.



# Aritmetične operacije

---

- $+$  - seštevanje
- $-$  - odštevanje
- $*$  - množenje
- $/$  - deljenje
- $\%$  - celoštevilčni ostanek





# Prireditveni stavek

---

- `int i,j,k;`
- `i = (2 + 3 ) * 10;`
- `i = 2 + 3 * 10;`
- `j = 6 % 4;`
- `k = i + j;`



# Podatkovni tipi

---

- Tip char – en znak
- Tip int – integer – celo število
- Tip float – real – realno število
- Tip double – še večji float



# Podatkovni tip char

---

- Podatkovni tip, ki predstavlja znak
- A je znak, medtem ko 7 je številka
- Podatkovni tip char je 8 biten
- ASCII tabela – enako kot pri pascalu

- Deklaracija

```
char imespremenljivke;
```

```
char imespremenljivke2;
```

```
char imespremenljivke3;
```

```
ali
```

```
char imespr1, imespr2, imespr3;
```



# Char - prireditev

---

- `char x;`
- `x = 'A';`
- `x = 65;`

Ali

- `x = 'a'`
- `x = 97;`



# Escape znaki

---

- `\n` – skok v novo vrstico
- `\b` – backspace – premakne kurzor v levo
- `\f` – form-feed – gremo na začetek strani
- `\r` – return znak – gremo na začetek vrstice
- `\t` – tabulator – enako kot da pritisnemo tab
- Znake ravno tako izpisujemo z `printf` funkcijo, le za znakovne spremenljivke uporabimo `%c` za izpis



# Izpis znakov – prireditev znak

---

```
#include <stdio.h>
int main(void)
{
    char c1;
    char c2;
    c1 = `A`;
    c2 = `a`;
    printf("Izpis vrednosti spremenljivke c1 %c.\n", c1);
    printf("Izpis vrednosti spremenljivke c2 %c.\n", c2);
    return 0;
}
```

# Izpis znakov - prireditev število

---

```
#include <stdio.h>
int main(void)
{
    char c1;
    char c2;
    c1 = 65;
    c2 = 97;
    printf("Izpis vrednosti spremenljivke c1 %c.\n", c1);
    printf("Izpis vrednosti spremenljivke c2 %c.\n", c2);
    return 0;
}
```



# Podatkovni tip int

---

- Večina sistemov, kjer se uporablja C je 32 bitnih (iz kjer C izvira) – torej celo število več ni omejeno na interval od 32767 do -32768 – 16bit, ampak na velikost 2147483647 do -2147483648





# Deklaracija int

---

- Deklaracija

```
int imespremenljivke;
```

```
int imespremenljivke2;
```

```
int imespremenljivke3;
```

```
ali
```

```
int imepr1, imespr2, imespr3;
```



# Izpis celih števil – pretvorba znakov v ASCII kodo

---

```
#include <stdio.h>
int main(void)
{
    char c1;
    char c2;
    c1 = 65;
    c2 = 97;
    printf("Izpis vrednosti spremenljivke c1 %d.\n", c1);
    printf("Izpis vrednosti spremenljivke c2 %d.\n", c2);
    return 0;
}
```



# Izpis celih števil

---

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int c1;
```

```
    int c2;
```

```
    c1 = 65; c2 = 97;
```

```
    printf("Vsota spremenljivk c1 in c2 = %d.\n", c1 + c2);
```

```
    return 0;
```

```
}
```



# Podatkovni tip float

---

- Podatkovni tip float v Cju je enakovreden podatkovnemu tipu real v pascalu.
- Vsebuje celo število + decimalni del



# Deklaracija float

---

- Deklaracija

```
float imespremenljivke;
```

```
float imespremenljivke2;
```

```
float imespremenljivke3;
```

```
ali
```

```
float imepr1, imespr2, imespr3;
```

- Pri izpisu uporabljamo %f za izpis spremenljivke



# Primer float

---

```
#include <stdio.h>
int main(void)
{
    float st1,st2,st3;
    st1 = 32 / 10;
    st2 = 32.0 / 10;
    st3 = 32 / 10.0;
    printf("Rezultat 32/10 is: %f\n", st1);
    printf("Rezultat 32.0/10 is: %f\n", st2);
    printf("Rezultat 32/10.0 is: %f\n", st3);
    return 0;
}
```



# Podatkovni tipi - nadaljevanje

---

- Nepredznačeni podatkovni tipi unsigned

```
int main(void)
```

```
{ unsigned int a;  
  unsigned int b; }
```

- Kratko in podaljšano celo število (byte, word v pascalu)

```
int main(void)
```

```
{ short int a;  
  long int b; }
```



# Posebnosti o imenu spremenljivke

---

- Spremenljivka ne sme vsebovati nobenih aritmetičnih operacij (+, -, ...)
- Spremenljivka ne sme vsebovati znaka .
- Spremenljivka ne sme vsebovati znaka `
- Spremenljivka ne sme vsebovati posebnih simbolov (npr \*, @, #, ?, ...)





# Branje podatkov

---

- Izpis z uporabo funkcije printf
- Obstajajo še naslednje funkcije
- `getc()`, `putc()`, `getchar()`, `putchar()`



# Branje podatkov iz tipkovnice

---

- Branje iz tipkovnice je še vedno najbolj uporaben
- `getc()` funkcija prebere znak in ga vrne kot celo število

```
#include <stdio.h>
```

```
int getc(FILE *stream);
```



# Standardni vhod in izhod - tok

---

- `stdio.h` nam definira tri tokove, katere lahko uporabimo
- `stdin` – standardni vhod za branje (tipkovnica)
- `stdout` – standardni izhod za pisanje (monitor)
- `stderr` – standardni izhod za napake (monitor)



# Primer branja znaka

---

```
#include <stdio.h>

int main(void)
{
    int ch;
    printf("Vnesi znak:\n");
    ch = getc(stdin);
    printf("Vnesen znak je: %c\n", ch);
    return 0;
}
```



# Branje iz tipkovnice #2

---

- C ima tudi funkcijo `getchar()`, ki je uporabljena za branje znakov. Funkcija nam naredi enako kot `getc()`

```
#include <stdio.h>
```

```
int getchar(void);
```

- Razlika je samo v tem, da nimamo vhodnega tokova – branje iz `stdin`



# Primer #2

---

```
#include <stdio.h>
int main(void)
{ int ch1, ch2;
  printf("\Vnesi dva znaka:\n");
  ch1 = getc( stdin );
  ch2 = getchar( );
  printf("\Prvi vnesen znak je: %c\n", ch1);
  printf("\Drugi vnesen znak je: %c\n", ch2);
  return 0;
}
```



# Izpis na zaslon

---

- Poleg funkcije printf imamo tudi funkciji putc in putchar. Funkciji sta enakovredni getc in getchar, le da izpisujeta na zalon (stdout ali stderr)

```
#include <stdio.h>
```

```
int putc(int c, FILE *stream);
```

```
int putchar(int c);
```



# Primer #3

---

```
#include <stdio.h>
int main(void)
{
    putchar(65);
    putchar(10);
    putchar(66);
    putchar(10);
    putchar(67);
    putchar(10);
    return 0;
}
```





# printf pod drobnogledom

---

- Prva funkcija, ki smo jo uporabili za izpis podatkov

```
#include <stdio.h>
```

```
int printf(const char *format-string, ...);
```

- `const char *format string` nam pomeni naš dejanski tekst, ki se izpiše.
- ... pa pomenijo naše spremenljivke. Vsaka spremenljivka mora imeti znotraj teksta svoj znak za izpis.



# Znaki za izpis spremenljivk

---

- %c – izpiše znak
- %d – izpiše celo desetiško število
- %i – izpiše celo število (enako kot %d)
- %f – izpiše realno število
- %o - izpiše število v osmiškem sestavu
- %s – izpiše niz



# Znaki za izpis spremenljivk

---

- %u – nepredznačeno celo število
- %x – število v šestnajstiškem sestavu (majhne črke)
- %X – število v šestnajstiškem sestavu (velike črke)
- %% - izpiše znak za procent (%)



# Maska pri izpisu

---

- Možnost rezerviranja prostora za izpis
- Številko vnesemo med % in d (za celo število) - %5d – rezerviramo 5 znakov za izpis
- Na ta način so števila poravnana na desno stran
- Če hočemo poravnavo na levo stran uporabimo predznak - - %-5d



# Maska pri izpisu #2

---

- Vnašanje ničel pred številko – uporabimo 0 med % in d - %05d
- Decimalna števila – hočemo decimano število na dve mesti natančno %0.2f



# Prireditveni stavki

---

- Veliko uporabljeni =
- $a = 5 * 6;$
- Tu se c ne vstavi:  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$
- Kaj pomeni kateri znak?



# Prireditveni stavki

---

$x += y;$  je enako  $x = x + y;$

$x -= y;$  je enako  $x = x - y;$

$x *= y;$  je enako  $x = x * y;$

$x /= y;$  je enako  $x = x / y;$

$x \% = y;$  je enako  $x = x \% y;$



# Prireditveni stavki – PAZI!

---

- $z = z * x + y;$

NI ENAKO KOT

- $z *= x + y;$

ker

- $z *= x + y$

je enako

- $z = z * (x + y);$





# Večanje in manjšanje za 1

---

- V pascalu poznamo inc in dec proceduri
- C pozna ukaza ++ (za povečanje) in – (za zmanjšanje)
- $x = x + 1;$  je enako kot  $x++;$
- $x = x - 1;$  je enako kot  $x--;$



# Večanje in manjšanje za 1 #2

---

- Obstajata dve možnosti za večanje in manjšanje za 1
- `x++` in `++x`;
- `x++` - število `x` se najprej priredi spremenljivki in nato se poveča
- `++x` - število `x` se najprej poveča in nato se šele priredi vrednost spremenljivki



# Večanje in manjšanje za 1 #3

---

- Primer

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int num1;
```

```
    num1 = 5;
```

```
    printf("stevilo %d \n",num1++); // izpise 5
```

```
    num1 = 5;
```

```
    printf("stevilo %d \n",++num1); // izpise 6
```

```
    return 0;
```

```
}
```



# Pogoji - operacije

---

- Enakost ==
- Različno !=
- Večje kot >
- Manjše kot <
- Večje ali enako >=
- Manjše ali enako <=
- Operatorji >, <, >=, in <= imajo večjo prioriteto kot == in !=



# Primeri

---

- $x * y < z + 3$ ; je enako kot  $(x * y) < (z + 3)$ ;
- Če  $x = 5$ ; in  $y = 6$ ; in primerjamo  $x < y$  nam primerjava vrne 1
- Torej rezultat je 1, ko je primerjava uspešna in 0, ko je primerjava neuspešna



# Primeri #2

---

```
#include <stdio.h>
int main(void)
{ int x, y;
  double z;
  x = 7; y = 25; z = 24.46;
  printf("x = %d, y = %d, in z = %.2f,\n", x, y, z);
  printf("x >= y primerjava: %d\n", x >= y);
  printf("x == y primerjava: %d\n", x == y);
  printf("x < z primerjava: %d\n", x < z);
  printf("y > z primerjava: %d\n", y > z);
  printf("x != y - 18 primerjava: %d\n", x != y - 18);
  printf("x + y != z primerjava: %d\n", x + y != z);
  return 0;
}
```



# Zanke

---

- Programski jezik C pozna naslednje zanke
- for
- while
- do-while



# for zanka

---

```
for (expression1; expression2; expression3)
{
    statement1;
    statement2;
    . . .
}
```





## for zanka #2

---

- expression1 – priprava spremenljivk – inicializacija spremenljivk
- expression2 – pogoj, kdaj se zanka konča
- expression3 – kako povečujemo števec
- statement1, statement2 – stavki, ki se izvajajo znotraj zanke



# Primer

---

```
#include <stdio.h>
int main(void)
{ int i;
  printf("Hex(velike) Hex(majhne) Decimalno\n");
  for (i=0; i<16; i++)
  {
    printf("%X %x %d\n", i, i, i);
  }
  return 0;
}
```



# Pazi

---

```
for (i=0; i<8; i++)  
    sum += i;
```

```
for (i=0; i<8; i++);  
    sum += i;
```

Ta dva stavka nista enaka! Prva zanka povečuje sum za i, druga zanka je prazna zanka (glej podpičje za zanko)!



# Prirejanje začetno vrednost več spremenljivkam

---

```
#include <stdio.h>

int main(void)
{int i, j;
    for (i=0, j=8; i<8; i++, j--)
        printf("%d + %d = %d\n", i, j, i+j);
    return 0;
}
```



# Večanje vrednosti več spremenljivkam

---

```
#include <stdio.h>

int main(void)
{ int i, j;
  for (i=0, j=1; i<8; i++, j++)
    printf("%d - %d = %d\n", j, i, j - i);
  return 0;
}
```



# Neskončna zanka

---

```
for ( ; ; )  
    { /* stavki */ }
```



# while zanka

---

```
while (expression)
{
    statement1;
    statement2;
    . . .
}
```



# while zanka

---

- Najprej se pogoj izvede in če je rezultat pogoja 1 ali več se stavki znotraj bloka izvršijo
- Zanka se ponavlja vse dokler pogoj ni 0





# Primer

---

```
#include <stdio.h>
int main(void)
{ int c;
  c = ` `;
  printf("\Vnesi znak:\n(vnesi x za izhod)\n");
  while (c != `x')
  {
    c = getc(stdin);
    putchar(c);
  }
  printf("\Konec zanke!\n");
  return 0;
}
```



# Neskončná zanka

---

```
while (1)
{
    statement1;
    statement2;
    . . .
}
```



# do-while zanka

---

```
do
{
    statement1;
    statement2;
    . . .
} while (expression);
```



# do-while zanka

---

- Najprej se izvedejo stavki znotraj bloka
- Nato se preverivrednost pogoja
- Če je vrednost pogoja različna od nič se zanka
- Zanka se ponavlja vse dokler pogoj ni 0



# Primer

---

```
#include <stdio.h>
int main(void)
{ int i;
  i = 65;
  do
  {
    printf("Vrednost %c je %d.\n", i, i);
    i++;
  } while (i<72);
  return 0;
}
```



# Logični operatorji

---

- Programski jezik C – ravno tako kot programski jezik pascal pozna logične operatorje
- AND - &&
- OR - ||
- NOT - !



# Logični operatorji

---

- AND

spr1	&&	spr2	rezultat
------	----	------	----------

0	&&	0	= 0
---	----	---	-----

0	&&	1	= 0
---	----	---	-----

1	&&	0	= 0
---	----	---	-----

1	&&	1	= 1
---	----	---	-----



# Logični operatorji

---

- OR

spr1		spr2	rezultat
------	--	------	----------

0		0	= 0
---	--	---	-----

0		1	= 1
---	--	---	-----

1		0	= 1
---	--	---	-----

1		1	= 1
---	--	---	-----





# Logični operatorji

---

- NOT

spr1	rezultat
------	----------

! 0	= 1
-----	-----

! 1	= 0
-----	-----



# IF stavek

---

- Če je pogoj izpolnjen se stavki izvedejo, če ni izpolnjen, se stavki preskočijo

if (pogoj)

{

stavek1;

stavek2;

}



# If-else stavek

---

- Če je pogoj izpolnjen se izvede prvi del stavka, drugače pa drugi del stavka

```
if (pogoj)
```

```
{
```

```
    stavek1;
```

```
}
```

```
else
```

```
{
```

```
    stavek2;
```

```
}
```



# Switch stavek

---

- Switch stavek je če primerjamo pascal ekvivalenten case stavku

switch (pogoj)

{

case 1: stavek1; break;

case 2: stavek2; break;

...

default: privzeti-stavek;

}



# Primer switch stavka

---

```
#include <stdio.h>

int main()
{ int dan;

  printf("Vnesi dan v tednu (znotraj
  intervala 1 do 7):\n");
  dan = getchar();
  switch (dan){
    case `1': printf("Dan 1\n"); break;
```



# Break in continue v zankah

---

- Break nam zanko prekine, medtem ko continue ukaz nam izvede naslednjo iteracijo v zanki (preskoči nadaljnje izvajanje trenutne iteracije)



# Break in for zanka

---

```
for (i=0; i < 10; i++)  
{  
    if (i == 7)  
        break;  
    printf("%d\n",i);  
}
```

- Zanka dejansko izpiše samo od 0 do 7



# Continue in for zanka

---

```
for (i=0; i < 10; i++)  
{  
    if (i == 7)  
        continue;  
    printf("%d\n",i);  
}
```

- Zanka nam izpiše števila od 0 – 9, toda število 7 se ne izpiše





# Nizi v programskem jeziku C

---

- Niz je dejansko polje znakov

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    char ime[20] = "Moje ime je Andrej.";
```

```
    printf("%s\n",ime);
```

```
}
```



# Nizovne funkcije

---

- Potrebno je vključiti datoteko string.h z include direktivo

```
#include <string.h>
```

- Dolžina niza

```
size_t strlen(const char *s);
```

- Funkcija nam vrne celo število – vrednost je število znakov v spremenljivki s



# Nizovne funkcije

---

```
char *strcpy(char *ponor, const char *izvor);
```

- Funkcija kopira iz spremenljivke izvor v spremenljivko ponor. Funkcija ob uspešnem kopiranju vrne kopirani niz



# Funkcije za branje/izpis nizov

---

```
char *gets(char *s);
```

- Funkcija nam prebere vrednost iz tipkovnice in shrani v spremenljivko s

```
int puts(const char *s);
```

- Funkcija nam izpiše vrednost iz spremenljivke s na zaslon lahko pa uporabimo tudi printf

```
printf("%s\n",s);
```



# Scanf funkcija

---

- Naprednejši način za branje iz tipkovnice je scanf funkcija
- Funkcija lahko prebere celo število, realno število, znak, niz, skratka vse podatkovne strukture

`int scanf(const char *format, ...);`

- Prvi del je opisni del (enako kot pri print), drugi del pa so nanizane spramenljivke z znakom & odspredaj



# Znak &

---

- Znak & pomeni, da vnašamo vrednosti v pomnilnik, kamor kaže spremenljivka



# Primer branja z scanf

---

```
#include <stdio.h>
Int main()
{
    char str[80]; int x, y; float z;
    printf("Vnesi dve celi števili (loči jih presledek):\n");
    scanf("%d %d", &x, &y);
    printf("Vnesi realno število:\n");
    scanf("%f", &z);
    printf("Vnesi niz:\n");
    scanf("%s", &str);
    printf("Ravnokar si vnesel:\n");
    printf("%d %d\n%f\n%s\n", x, y, z, str);
    return 0;
}
```



# Imena funkcij

---

- Ime funkcije se ne sme začeti z številko
- Ime funkcije se ne sme začeti z  $*$
- Ime funkcije se ne sme začeti z  $+$  (ali drugim aritmetičnim znakom)
- Ime funkcije se ne sme začeti z  $.$
- Ime funkcije ne sme vsebovati znaka  $-$
- Ime funkcije ne sme vsebovati znaka  $`$





# Primeri imena funkcij

---

- print2copy
- total\_number
- \_quick\_add
- method3



# Izdelava lastnih funkcij

---

```
int seštej(int x, int y)
{
    int res;
    res = x + y;
    return res;
}
```



# Uporaba lastnih funkcij

---

```
#include <stdio.h>
int sestej(int x,int y)
{
    int res;
    res = x + y;
    return res;
}
```

```
int main(void)
{
    int vsota;
    vsota = sestej(5,6);
    printf("Vsota stevil 5 in 6 je %d.\n",vsota);
    return 0;
}
```



# Vnos parametrov po vrednosti

---

- Prejšnja funkcija je bila izdelana, da se parametri vnašajo po vrednosti (poznamo iz pascala)

```
int sestej(int x,int y)
```

```
{
```

```
    int res;
```

```
    res = x + y;
```

```
    x = 6;
```

```
    y = 10;
```

```
    return res;
```

```
}
```

- Vrednost x in y se po končani funkciji sestej postavijo na prvotno vsebino (vrednosti x in y se izgubijo)



# Vnos parametrov po referenci

---

- Izdelajmo funkcijo, da se vrednosti po končanem izvajanju ne izgubijo

```
int sestej(int *x,int *y)
```

```
{
```

```
    int res;
```

```
    res = *x + *y;
```

```
    *x = 6;
```

```
    *y = 10;
```

```
    return res;
```

```
}
```

- Funkcija na ta način od nas zahteva, da vnesemo dejansko naslov spremenljivke in se vrednost po izteku funkcije tudi ohranijo



# Primer

```
#include <stdio.h>
int sestej(int *x,int *y)
{
    int res;
    *x = 6;*y = 10;
    return *x + *y;
}
int sestej2(int x, int y)
{
    int res;
    res = x + y; x = 6; y = 10;
    return res;
}
int main(void)
{
    char ime[20];
    int a,b; a = 2; b = 3;
    printf("%d + %d = %d\n",a,b,sestej2(a,b));
    printf("Vrednosti po %d + %d = %d\n",a,b,sestej(&a,&b));
    scanf("%s",ime);
    return 0;
}
```



# Dodatne funkcije za delo z nizi

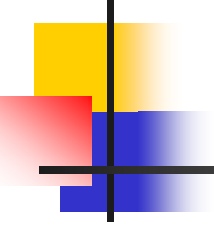
---

```
char *strncpy(char *dest, const char *src, size_t n);
```

```
int  strcasecmp(const char *, const char *);
```

```
int strcmp(const char *s1, const char *s2);
```

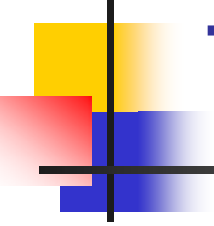
```
int strncmp(const char *s1, const char *s2, size_t n);
```



# Dvodimenzionalne tabelle

---





# Trodimensionálne tabele

---