Uvod u programiranje
- predavanja -

listopad 2025.

10. Tipovi podataka u programskom jeziku C

Vodič „09 prije desetog predavanja”

- Očekuje se da ste savladali (na razini na kojoj je obrađeno u vodiču):
  - double
  - long double

- Pitanja?

Numeričke pogreške

- Neki dekadski brojevi se ne mogu prikazati pomoću konačnog broja binarnih znamenaka
  - Primjer:

```
float f = 0.3f;
printf("%18.16f", f);
```

```
0.3000000119209290
```

0011 1110 1001 1001 1001 1001 1001 1001 1001 1001 ...

0011 1110 1001 1001 1001 1001 1001 1010 (zaokruženo na bližu vrijednost)

- Neki realni brojevi imaju konačan broj, ali ipak "previše" binarnih znamenaka
  - Primjer:

```
float f = 4194304.125f;
printf("%14.6f", f);
```

```
4194304.000000
```

0100 1010 1000 0000 0000 0000 0000 0000 01

0100 1010 1000 0000 0000 0000 0000 0000 (zaokruženo na bližu vrijednost)

Numeričke pogreške

- Računanje s brojevima bitno različitog reda veličine može dovesti do numeričke pogreške
  - Primjer

```
float f = 6000000.0f; float malif = 0.25f;
f = f + malif;
f = f + malif;
f = f + malif;
f = f + malif;
printf("%f ", f);
```

```
6000000.000000
```

  - bolje:

```
float malaSuma = 0.f;
malaSuma = malaSuma + malif;
malaSuma = malaSuma + malif;
malaSuma = malaSuma + malif;
malaSuma = malaSuma + malif
f = f + malaSuma;
printf("%f ", f);
```

```
6000001.000000
```

Numeričke pogreške

- Potreban je oprez kod uspoređivanja realnih vrijednosti
  - jesu li dva broja "jednaka"
    - možda treba uzeti u obzir neku toleranciju
  - kada je kontrolna varijabla petlje dosegla granicu za prekid petlje?

Primjer

- Učitati pozitivan cijeli broj n (nije potrebno provjeravati je li učitan ispravan broj). Zatim ispisati realne brojeve od 0.5 do n s korakom od 0.1. Za ispis realnih brojeva koristiti formatsku specifikaciju %12.9f.

```
Upisite n > 2↵
 0.500000000↵
 0.600000000↵
 0.700000000↵
 0.800000000↵
 0.900000000↵
 1.000000000↵
 1.100000000↵
 1.200000000↵
 1.300000000↵
 1.400000000↵
 1.500000000↵
 1.600000000↵
 1.700000000↵
 1.800000000↵
 1.900000000↵
 2.000000000↵
```

Primjer

```
#include <stdio.h>

int main(void) {
   int n;
   float x;
   printf("Upisite n > ");
   scanf("%d", &n);

   for (x = 0.5f; x <= n; x = x + 0.1f) {
      printf("%12.9f\n", x);
   }
}
```

```
Upisite n > 2↵
 0.500000000↵
 0.600000024↵
 0.700000048↵
 0.800000072↵
 0.900000095↵
 1.000000119↵
 1.100000143↵
 1.200000167↵
 1.300000191↵
 1.400000215↵
 1.500000238↵
 1.600000262↵
 1.700000286↵
 1.800000310↵
 1.900000334↵
```

- zašto se ne ispisuju točne vrijednosti?
  - čak niti za 1.0 ili 1.5?
- zašto se petlja zaustavila prerano?

Primjer

- Poboljšanje:
  - dopustiti određenu toleranciju kod uspoređivanja brojeva/uvjeta za prekid petlje

```
#include <stdio.h>
#define TOLER 0.00001f

int main(void) {
   int n;
   float x;
   printf("Upisite n > ");
   scanf("%d", &n);

   for (x = 0.5f; x <= n + TOLER; x = x + 0.1f) {
      printf("%12.9f\n", x);
   }
}
```

```
Upisite n > 2↵
 0.500000000↵
 0.600000024↵
 0.700000048↵
 0.800000072↵
 0.900000095↵
 1.000000119↵
 1.100000143↵
 1.200000167↵
 1.300000191↵
 1.400000215↵
 1.500000238↵
 1.600000262↵
 1.700000286↵
 1.800000310↵
 1.900000334↵
 2.000000238↵
```

Primjer

- Poboljšanje:
  - gdje je moguće, koristiti cijele brojeve
    - u operacijama s cijelim brojevima nema numeričkih pogrešaka

```
#include <stdio.h>

int main(void) {
   int n, brojac;
   float x;
   printf("Upisite n > ");
   scanf("%d", &n);

   for (brojac = 5;
        brojac <= n * 10;
        brojac = brojac + 1) {
      x = brojac / 10.f;
      printf("%12.9f\n", x);
   }
}
```

```
Upisite n > 2↵
 0.500000000
 0.600000024
 0.699999988
 0.800000012
 0.899999976
 1.000000000
 1.100000024
 1.200000048
 1.299999952
 1.399999976
 1.500000000
 1.600000024
 1.700000048
 1.799999952
 1.899999976
 2.000000000
```

Primjer

- Objasniti razlike u rezultatima sljedećih operacija:

```
double x;

x = 0.1f + 0.1f;
printf("%20.18f\n", x);

x = 0.1f + 0.1;
printf("%20.18f\n", x);

x = 0.1 + 0.1;
printf("%20.18f\n", x);
```

```
0.200000002980232240
0.200000001490116120
0.200000000000000010
```

Definicija tipa podataka

typedef

Definicija tipa

- kreiranje zamjenskog imena (sinonim) za postojeći tip podataka
  - unapređenje razumljivosti (dokumentiranosti) programskog koda
  - unapređenje prenosivosti (portabilnosti) programskog koda
  - pojednostavljivanje naredbi za definiciju varijabli
- opći oblik naredbe

  `typedef nazivPostojecegTipa noviNazivTipa;`
  - naredba se mora nalaziti ispred (ne nužno neposredno) naredbi koje će koristiti novi naziv tipa
- Primjer:

```
int main(void) {
   typedef int cijeliBroj_t;      Ovo je samo ilustracija.
   typedef float realniBroj_t;    Ne definirati nove tipove samo radi prijevoda!
   cijeliBroj_t m, n;
   ...
   realniBroj_t x, y;
```

Unapređenje razumljivosti (dokumentacija)

- postojećem tipu podataka dodati zamjensko ime
  - dodatno opisuje domenu vrijednosti varijabli

```
typedef unsigned int maticniBroj_t;
typedef unsigned char ocjenaBroj_t;
typedef char ocjenaSlovo_t;
...
maticniBroj_t mbrStud1 = 123456789;
ocjenaBroj_t ocjeneGrupe[10] = {1, 1, 5, 4, 2, 1, 1, 4, 3, 5};
ocjenaSlovo_t najmanjaSlovnaOcjena = 'F';
```

pretpostavlja se da mbr. nema više od 9 znamenaka i da se za pohranu tipa int koristi 4 bajta

konvencija (nije pravilo): dodati _t na kraj imena

razumljivije je od

```
unsigned int mbrStud1 = 123456789;
unsigned char ocjeneGrupe[10] = {1, 1, 5, 4, 2, 1, 1, 4, 3, 5};
char najmanjaSlovnaOcjena = 'F';
```

Unapređenje prenosivosti

- ako prethodni program treba prenijeti na platformu na kojoj je tip podatka int pohranjen u samo dva bajta [0, 65 535], dok je long int pohranjen u 4 bajta [0, 4 294 967 295]
  - umjesto unsigned int za sve podatke o matičnim brojevima trebat će koristiti unsigned long int
  - ako se u programu koristilo zamjensko ime tipa maticniBroj_t, dovoljno je (na samo jednom mjestu!) promijeniti definiciju tipa i prevesti program na novoj platformi
    - inače, trebalo bi promijeniti tip u definiciji svih varijabli u kojima se pohranjuje matični broj

```
...
typedef unsigned int maticniBroj_t;
typedef unsigned long int maticniBroj_t;
...
```

Prenosivost - ugrađene definicije tipova

- neka zamjenska imena za tipove unaprijed su definirana
  - npr. operator sizeof (koji će detaljnije biti objašnjen kasnije) vraća veličinu objekta (npr. varijable) izraženu u bajtovima
  - radi se o cijelom broju koji na različitim platformama može imati različite raspone i biti definiran na temelju različitih osnovnih tipova
    - npr. unsigned int ili unsigned long ili unsigned long long
  - u stdlib.h je definiran tip size_t koji se u većini slučajeva može na siguran način koristiti kao da se radi o tipu unsigned int

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
   int n;
   size_t velicina;
   velicina = sizeof(n);
   printf("%u", velicina);
   ...
```

4

Prenosivost - ugrađene definicije tipova

- broj bajtova koji se koristi za pohranu cjelobrojnih tipova podataka u C-u nije propisan jezikom
  - rasponi se razlikuju u različitim prevodiocima i arhitekturama
  - ako je potrebno fiksirati preciznost i raspon, neovisno o prevodiocu i arhitekturi

```
#include <stdio.h>
#include <stdint.h>
int main(void) {
   int m;                    veličina (dakle i raspon) ovisi o prevodiocu
   int8_t n1;                točno 1 bajt, signed
   int16_t n2;               točno 2 bajta, signed
   int32_t n4;               točno 4 bajta, signed
   int64_t n8;               točno 8 bajtova, signed
   uint8_t un1;              točno 1 bajt, unsigned
   uint16_t un2;             točno 2 bajta, unsigned
   ...
```

Pojednostavljivanje naredbi za definiciju

- definicija tipa često se koristi umjesto deklaracije strukture

s pomoću deklaracije strukture:

```
struct datum_s {
   int dan;
   int mj;
   int god;
};
struct interval_s {
   struct datum_s dat_od;
   struct datum_s dat_do;
};

struct interval_s zim_rok =
  {{11, 2, 2019}, {22, 2, 2019}};

struct interval_s ljetni_rok =
  {{17, 6, 2019}, {3, 7, 2019}};
```

s pomoću definicije tipa:

```
typedef struct {
   int dan;
   int mj;
   int god;
} datum_t;
typedef struct {
   datum_t dat_od;
   datum_t dat_do;
} interval_t;

interval_t zim_rok =
  {{11, 2, 2019}, {22, 2, 2019}};

interval_t ljetni_rok =
  {{17, 6, 2019}, {3, 7, 2019}};
```

Zadatak

Caesar cipher

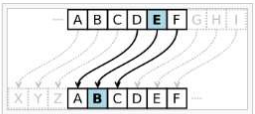
56 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

In cryptography, a **Caesar cipher**, also known as **Caesar's cipher**, the **shift cipher**, **Caesar's code**, or **Caesar shift**, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence.

The encryption step performed by a Caesar cipher is often incorporated as part of more complex schemes, such as the Vigenère cipher, and still has modern application in the ROT13 system. As with all single-alphabet substitution ciphers, the Caesar cipher is easily broken and in modern practice offers essentially no communications security.



The action of a Caesar cipher is to replace each plaintext letter with a different one a fixed number of places down the alphabet. The cipher illustrated here uses a left shift of 3, so that (for example) each occurrence of E in the plaintext becomes B in the ciphertext.

Example [edit]

The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places, equivalent to a right shift of 23 (the shift parameter is used as the key):

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

When encrypting, a person looks up each letter of the message in the "plain" line and writes down the corresponding letter in the "cipher" line.

```
Plaintext:  THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
```

https://en.wikipedia.org/wiki/Caesar_cipher

- **Napišite program koji će učitati ključ (pomak, shift) i:**
  - Šifrirati znakovni niz „Hello world!” Cezarovom šifrom
  - Potom dešifrirati šifrirani niz.

```
Unesite kljuc (pomak): 1
Plain:   Hello world!
Coded:   Ifmmp!xpsme"
Decoded: Hello world!
```

Izrazi s različitim tipovima podataka

Implicitna pretvorba tipova podataka

Izrazi s različitim tipovima podataka

- Što je rezultat operacije `i + f`
  - `2. + 2.99` ⇒ `4.99` ili `2 + 2` ⇒ `4`
- Što je rezultat operacije `i >= f`
  - `2. >= 2.99` ⇒ `0` ili `2 >= 2` ⇒ `1`

```
int i;
float f;
i = 2;
f = 2.99f;
```

- Kada su operandi različitog tipa, prije obavljanja operacije obavlja se implicitna (automatska) pretvorba (konverzija) "manje važnog" tipa operanda u "važniji" od tipova operanada koji sudjeluju u operaciji.
- U prikazanim primjerima, prije nego se obavi operacija, vrijednost koja se nalazi u varijabli `i` (int je "manje važan") pretvara se u vrijednost `2.f` (float je "važniji" tip)
  - pri tome tip i sadržaj varijable `i` ostaje nepromijenjen.

Implicitna pretvorba tipova podataka

- Implicitna pretvorba tipova podataka u aritmetičkim i relacijskim izrazima obavlja se prema jednom od sljedećih 6 pravila. Treba iskoristiti prvo po redu pravilo koje se može primijeniti na konkretan slučaj!

  1. Ako je jedan od operanada tipa **long double**, preostali operand se pretvara u tip **long double**
  2. Ako je jedan od operanada tipa **double**, preostali operand se pretvara u tip **double**
  3. Ako je jedan od operanada tipa **float**, preostali operand se pretvara u tip **float**
  4. Ako je jedan od operanada tipa **long long**, preostali operand se pretvara u tip **long long**
  5. Ako je jedan od operanada tipa **long**, preostali operand se pretvara u tip **long**
  6. Operandi tipa **_Bool**, **short** i **char** pretvaraju se u tip **int**

- Kada se u izrazima pojavljuju unsigned tipovi, pravila pretvorbe su složenija. Zato se ovdje neće razmatrati.

Primjer

```
_Bool b;        char c;         short s;
int i;          long l;         long long ll;
float f;        double d;       long double ld;
```

Izraz	Pretvorba tipa prije obavljanja operacije	Prema pravilu
`ld * i`	sadržaj **i** → long double	1.
`c % i`	sadržaj **c** → int	6.
`s / f`	sadržaj **s** → float	3.
`l % i`	sadržaj **i** → long	5.
`d + c`	sadržaj **c** → double	2.
`s - c`	sadržaj **s** → int, sadržaj **c** → int	6.
`b == ll`	sadržaj **b** → long long	4.

Pretvorba tipova podataka kod pridruživanja

- Kod operacije pridruživanja, podatak s desne strane operatora pridruživanja pretvara se u tip podatka lijeve strane izraza
  - Primjer:

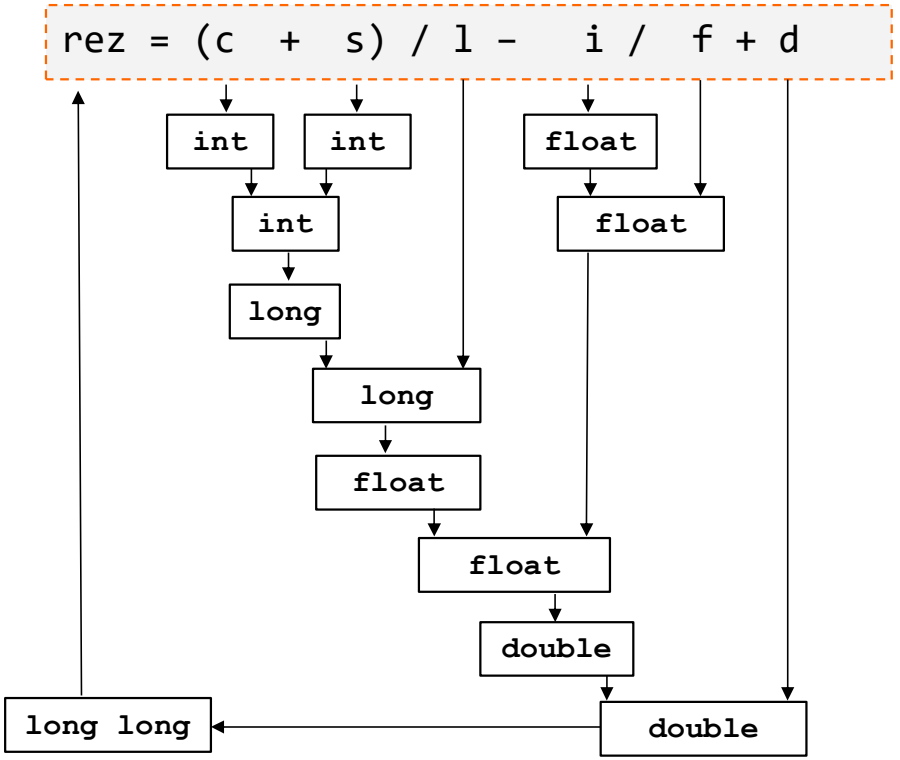
```
int i;
char c = '2';
long double ld;

i = 1.75;
ld = i + c;
```

  - nakon inicijalizacije varijabli
    - vrijednost konstante tipa double (1.75) pretvara se u vrijednost tipa int (1) i pridružuje varijabli i
    - vrijednost varijable c tipa char (50 u jednom bajtu) pretvara se u int (50 u četiri bajta), izračunava se rezultat tipa int (51) , pretvara se u long double (51.L) i pohranjuje u varijablu ld

Primjer

```
char c;
short int s;
int i;
long l;
float f;
double d;
long long rez;
```



Eksplicitna (zadana) pretvorba tipa podataka

- Opći oblik zadane (eksplicitne) pretvorbe (eng. cast operator):

  `(tip_podatka) izraz`

- Primjer: radi realnog dijeljenja dviju cjelobrojnih varijabli, korištena je nespretna pomoćna operacija množenja realnom konstantom

```
int i, j;
float x;
...
x = 1.f * i / j;
```

  - operator *cast* omogućava puno bolje rješenje:

```
x = (float) i / j;
```

  - analizirajte zašto sljedeće nije ispravno:

```
x = (float) (i / j);
```

Eksplicitna (zadana) pretvorba tipa podataka

- Operator *cast* ne treba primjenjivati bez potrebe
  - Umjesto korištenja operatora za eksplicitnu pretvorbu tipa nad konstantom, koristiti konstantu odgovarajućeg tipa

  - Primjer: ako se u realnu varijablu x želi pridružiti vrijednost ⅓

```
float x;

x = 1 / 3;              neispravno! U varijablu x će se upisati vrijednost 0.0
x = (float) 1 / 3;      nepotrebno! Nema potrebe za operatorom cast
x = 1.f / 3;            ispravno
x = 1 / 3.f;            ispravno
x = 1.f / 3.f;          ispravno
```

Pretvorba tipova podataka - mogući gubici

- Potrebno je obratiti pozornost da se pri pretvorbi jednog u drugi tip podatka može "izgubiti" manji ili veći dio informacije

```
int i;
float x;
i = 2.99999f;
x = 2147483638;
printf("%d\n%f", i, x);
```

```
2
2147483648.000000
```

  - "izgubljene" su decimale 0.99999 kod pretvorbe vrijednosti
  - float tip ima premalo bitova mantise za točan prikaz vrijednosti. Najbliži broj koji se može prikazati je za 10 veći od 2147483638

Primjer

```
int i = 2147000000;
float x;
x = i;
printf("%f\n", x);
x = x + 1.f;
printf("%f\n", x);
```

```
2147000064.000000
2147000064.000000
```

- pri operaciji pridruživanja x = i, cjelobrojna vrijednost 2147000000 pretvorila se u tip podatka float. Najbliža realna vrijednost koja se može prikazati u standardnoj preciznosti je 2147000064.0
- tijekom zbrajanja vrijednosti 2147000064.0 i 1.0, došlo je do numeričke pogreške zbog zbrajanja vrlo velike i vrlo male vrijednosti, rezultat zbrajanja je 2147000064.0 te se ta vrijednost pridružila varijabli x
- obje pogreške posljedica su ograničenog broja bitova mantise

Primjer

```
double d1 = 123456789.;
double d2 = 1.e40;
float f1 = d1;
float f2 = d2;
printf("%lf\n%f\n"
       "%lf\n%f",
        d1, f1,
        d2, f2);
```

```
123456789.000000
123456792.000000
10000000000000003037860284270036668907 52.000000
inf
```

- nedovoljno bitova mantise u f1 za točnu pohranu broja 123456789
- nedovoljno bitova mantise u d2 za točnu pohranu broja $1 \cdot 10^{40}$
- nedovoljni raspon (karakteristika) u f2 za pohranu broja $1 \cdot 10^{40}$

Primjer

- Koje su vrijednosti i tipovi rezultata evaluacije sljedećih izraza:

1. `3 / 2 * 2`
2. `3 / 2 * 2.`
3. `3 / 2. * 2`
4. `3 / (float)2 * 2`
5. `(double)3 / 2`
6. `(double)(3 / 2)`
7. `2+0.5   * 4`
8. `(2 + 0.5) * 4`
9. `(int)(0.5 + 2) * 4`
10. `(float) ((int)1.5 + 2) / 4`
11. `(int)1.6 + (int)1.6`
12. `(int)(1.6 + 1.6)`

Zadatak

KONTROLA OIB-a
(ISO7064, MOD 11,10 – Hibridni sistem)

1. **OIB ima 11 znamenaka od koji je posljednja tj. 11. znamenka kontrolna znamenka - dobivena je izračunom iz prethodnih 10 znamenaka po međunarodnoj normi ISO 7064 (MOD 11, 10).**

2. **Kontrolna znamenka prema navedenoj normi dobiva se slijedećim postupkom :**
   1. ***prva znamenka zbroji se s brojem 10***
   2. ***dobiveni zbroj cjelobrojno (s ostatkom) podijeli se brojem 10; ako je dobiveni ostatak 0 zamijeni se brojem 10 (ovaj broj je tzv. međuostatak)***
   3. ***dobiveni međuostatak pomnoži se brojem 2***
   4. ***dobiveni umnožak cjelobrojno (s ostatkom) podijeli se brojem 11; ovaj ostatak matematički nikako ne može biti 0 jer je rezultat prethodnog koraka uvijek paran broj***
   5. ***slijedeća znamenka zbroji se s ostatkom u prethodnom koraku***
   6. ***ponavljaju se koraci 2, 3, 4 i 5 dok se ne potroše sve znamenke***
   7. ***razlika između broja 11 i ostatka u zadnjem koraku je kontrolna znamenka; ako je ostatak 1 kontrolna znamenka je 0 (11-1=10, a 10 ima dvije znamenke)***

Primjer za OIB (provjera ispravnosti OIB-a : 69435151530)

korak	znamenka	zbroj broja iz stupca b s ostatkom iz prethodnog koraka (brojem iz stupca f) ili s brojem 10 za korak 1	ostatak od dijeljejnja s 10 broja iz stupca c – ako je 0 stavi se 10	umnožak broja iz stupca d s 2	ostatak od dijeljanja s 11 broja iz stupca e
a	b	c	d	e	f
1	6	16	6	12	1
2	9	10	10	20	9
3	4	13	3	6	6
4	3	9	9	18	7
5	5	12	2	4	4
6	1	5	5	10	10
7	5	15	5	10	10
8	1	11	1	2	2
9	5	7	7	14	3
10	3	6	6	12	1

11-1=10; kontrolna znamenka je 0

https://regos.hr/app/uploads/2018/07/KONTROLA-OIB-a.pdf

- **Napišite program koji će učitati OIB i provjeriti ispravnost**

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **nema** ☺
  - **10. vježbe uz predavanja**