Programiranje i programsko inženjerstvo

Predavanja

2013. / 2014.

4. Ostali operatori

(Definicija varijable s inicijalizacijom)

Definicija varijable s inicijalizacijom

- Početna vrijednost varijable može se zadati istodobno s definicijom varijable

```
int m = 5, n = 7, k;
float f = 15.1f;
```

Varijablama **m**, **n**, **f** pridružene su vrijednosti **5**, **7** i **15.1f**. Varijabla **k** sadrži nedefiniranu (nepoznatu) vrijednost. U nastavku je prikazan (po rezultatu) ekvivalentan odsječak programa:

```
int m, n, k;
float f;
m = 5;
n = 7;
f = 15.1.f;
```

Ostali operatori

Unarni operatori

Operator	Značenje	Primjer
`sizeof`	zauzeće memorije	`sizeof(int)`
`(tip)`	*pretvorba tipa (cast)*	`(double) i`
`!`	*logičko ne*	`!( 5 > 2 )`
`+`	unarni plus	`+4 / 2`
`-`	unarni minus	`-5 * 3`
`++`	uvećavanje za 1	`++i` ili `k++`
`--`	umanjenje za 1	`--j` ili `k--`
`~`	inverzija bitova	`int x = ~0xFFFF;`
	→ *u poglavlju o operatorima nad bitovima*	
`*`	indirekcija	`*p`
`&`	adresni operator	`scanf("%d",&n);`
	→ *u poglavlju o pokazivačima*	

Unarni + i -

```
float x = 5.0f, y = 2.0f;
        + x + y          →  7.0f
        +x + +y          →  7.0f
        +x + + y         →  7.0f
        - x + - y        → -7.0f
        -x - -y          → -3.0f
        - x - - y        → -3.0f
```

+x NIJE apsolutna vrijednost od x

```
x = -5.0f;
        +x               →  -5.0f
```

Vježba: izračunavanje rješenja (korijena) kvadratne jednadžbe

- Napomene uz program:
  - Prva naredba **`#include <stdio.h>`** "uključuje" datoteku zaglavlja (*header*) s opisom funkcija i konstanti koje se odnose na rad s ulazom/izlazom (čitanje s tipkovnice i ispis na zaslon)
  - Druga naredba **`#include <math.h>`** "uključuje" datoteku zaglavlja s opisom matematičkih funkcija i konstanti
  - Za izračunavanje drugog korijena koristi se funkcija **`pow`** koja računa x^y , a koja se poziva na sljedeći način:
    `korijen=pow(x, y);`

Rješenje I dio

```
#include <stdio.h>
#include <math.h>
int main(void) {
   float a, b, c, x1, x2, q, d, x1r, x2r, x1i, x2i;
   printf("Zadajte koeficijente kvadratne jednadzbe a,b,c:");
   scanf("%f %f %f", &a, &b, &c);

   d = b*b - 4.0f*a*c; /* diskriminanta */
   if (d > 0) {
       /* Rjesenja su realna  */
       q = pow (d, 1./2);
       x1 = (-b + q)/(2*a);
       x2 = (-b - q)/(2*a);
       printf ("X1=%f   X2=%f\n", x1, x2);
```

Rješenje II dio

```
  } else if (d == 0) {
       /* postoji samo jedno rjesenje */
       x1 = -b/(2*a);
       printf ("X=%f\n", x1);
  } else {
       /* Rjesenja su konjugirano kompleksni broj   */
       q = pow(-d, 1.f/2);
       x1r = -b/(2*a) ;
       x2r = x1r;
       x1i = q/(2*a);
       x2i = -x1i;
       printf ("X1 = (%f,%f)\n", x1r, x1i);
       printf ("X2 = (%f,%f)\n", x2r, x2i);
  }
   return 0;
}
```

Operatori povećanja i smanjenja za 1

Jedan od razloga njihovog postojanja je djelotvornije izvršavanje (postoje posebne procesorske instrukcije za vrlo brzo obavljanje tih operacija):

Operator povećanja za 1

```
int m = 7;
float x = -15.2f;
++m;          ili    m++;
++x;          ili    x++;
printf("%d  %f\n", m, x);        → 8  -14.2f
```

Operator smanjenja za 1

```
int m = 7;
float x = -15.2f;
--m;          ili    m--;
--x;          ili    x--;
printf("%d  %f\n", m, x);        → 6  -16.2f
```

Operatori povećanja i smanjenja za 1

- prefiksni oblik (operator ispred varijable)
  `++a; --a;`
- postfiksni oblik (operator iza varijable)
  `a++; a--;`
- u oba oblika varijabla **a** se uvećava/umanjuje za 1, ali:

Prefiksni operator: u izrazima se vrijednost varijable prvo uveća/umanji, a tek onda "iskoristi" njena vrijednost

```
int i, j;
i = 2;
j = ++i;
```

i	j
2	?
3	3

```
int i, j;
i = 2;
j = --i;
```

i	j
2	?
1	1

Operatori povećanja i smanjenja za 1

Postfiksni operator: u izrazima se vrijednost varijable prvo "iskoristi", a nakon toga se varijabla uveća/umanji

```
int i, j;
i = 2;
j = i++;
```

i	j
2	?
3	2

```
int i, j;
i = 2;
j = i--;
```

i	j
2	?
1	2

Nema garancije da će se varijabla uvećati/umanjiti ODMAH nakon što se "iskoristi" njena vrijednost, ali se garantira da će se varijabla uvećati/umanjiti prije nego započne izvršavanje sljedeće naredbe (vidjeti primjer: *doing too much at once*).

Operatori povećanja i smanjenja za 1

- **Primjer:**

```
a = 5;
b = ++a * 2;
c = b++;
```

- Nakon izvođenja ovih naredbi
  - a ima vrijednost 6
  - b ima vrijednost 13
  - c ima vrijednost 12

Operatori povećanja i smanjenja za 1

- Izbjegavati dvoznačnosti oblika:

```
int i = 7;
i = 2 * i++;
```
(*doing too much at once*)

- ako se ++ obavi nakon pridruživanja, rezultat je 15
- ako se ++ obavi prije pridruživanja, rezultat je 14
- **Savjet:** izbjegavati dvije ili više promjena sadržaja iste varijable unutar iste naredbe. Još jedan loš primjer:

```
k = ++i * --i;
```

- Sljedeća naredba ne uzrokuje takve probleme:

```
k = m++ + n--;
```

- **Nije dopušteno:**

```
i = ++5;
m = (i+j)--;
```

Binarni operatori

Operator	*Značenje*	*Primjer*
`* / %`	množenje, dijeljenje	
`+ -`	zbrajanje, oduzimanje	
`<<`	pomak bitova u lijevo	`n = n << 2;`
`>>`	pomak bitova u desno	`n = n >> 1;`
`< > <= >=`	relacijski operatori	
`== !=`	operatori jednakosti	
`&`	logički I po bitovima	`znam & '0'`
`^`	isključivi ILI po bitovima	`f = f ^ f`
`|`	uključivi ILI po bitovima	`f = f | 0x80`
`&& ||`	logički I i ILI	

Operatori nad bitovima

- Operatori **&, |, ^** su binarni, a odnose se na usporedbu bitova dvaju operanada

&	AND
|	OR
^	XOR

b1	b2	b1 & b2	b1 | b2	b1 ^ b2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- Operator **~** je unarni, a odnosi se na operand s desna

~	NOT

b1	~b1
0	1
1	0

- Operatori posmaka bitova su binarni, a odnose se na posmak bitova lijevog operanda za iznos određen desnim operandom

<<	SHIFT LEFT
>>	SHIFT RIGHT

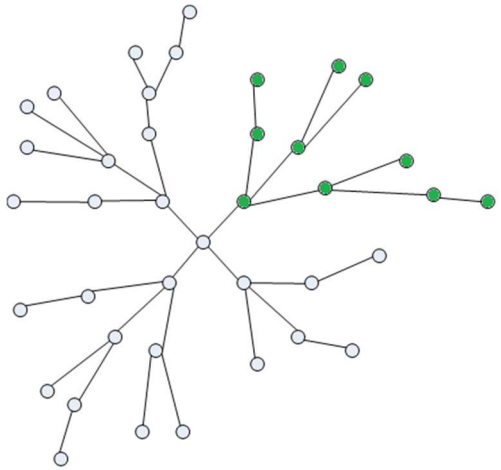
Primjer operacija s bitovima

- Izračunati izraze: **`10 & 7, 3 | 5, 3 ^ 5, ~3`**

```
  0000 0000 0000 0000 0000 0000 0000 1010 (10)
& 0000 0000 0000 0000 0000 0000 0000 0111  (7)
  -----------------------------------
  0000 0000 0000 0000 0000 0000 0000 0010  (2)

  0000 0000 0000 0000 0000 0000 0000 0011  (3)
| 0000 0000 0000 0000 0000 0000 0000 0101  (5)
  -----------------------------------
  0000 0000 0000 0000 0000 0000 0000 0111  (7)

  0000 0000 0000 0000 0000 0000 0000 0011  (3)
^ 0000 0000 0000 0000 0000 0000 0000 0101  (5)
  -----------------------------------
  0000 0000 0000 0000 0000 0000 0000 0110  (6)

~ 0000 0000 0000 0000 0000 0000 0000 0011  (3)
  -----------------------------------
  1111 1111 1111 1111 1111 1111 1111 1100 (–4)
```

Primjer operacija s bitovima

- **Podmreža *(subnet)*:**
  logička podmreža IP mreže



- IP adresa (v4) je 32-bitni broj
- Podmreža se definira mrežnim **prefiksom** - određenim brojem najznačajnijih bitova, npr. s prvih 25 bita
- IP adresa pripada podmreži ako ima isti prefiks kao i podmreža odnosno ako su im najznačajnijih N bitova jednaki
- Usporedba prefiksa se elegantno obavlja **bitovnom I operacijom**

Primjer operacija s bitovima

- Izračunajmo pripadaju li adrese:

  adr_1 = 11000000 10101000 00000111 01000110

  adr_2 = 11000000 10101000 00000101 10001100

  podmreži: **11000000 10101000 00000101 10000000** ?

  25

- U C-u:

```
int adr1, adr2, podmreza;
adr1 = 0xC0A80746;
adr2 = 0xC0A8058C;
podmreza = 0xC0A80580;
```

- U C-u ne postoji operator kojim se može usporediti prvih 25 bitova dviju cjelobrojnih varijabli, stoga se prvo određuje "maska koja propušta" prvih 25 bitova (a ostale bitove "maskira"):

  maska = 11111111 11111111 11111111 10000000

  U C-u: **maska = 0xFFFFFF80;**

  25

Primjer operacija s bitovima

- Zatim se "maskiraju" bitovi adrese te usporedi rezultat s podmrežom:

```
   11000000 10101000 00000111 01000110
 & 11111111 11111111 11111111 10000000
  ----------------------------------
   11000000 10101000 00000111 00000000
   11000000 10101000 00000101 10000000 ?
```

→nisu jednaki

U C-u: `if ((adr1 & maska) == podmreza) { ...`

- Isto za drugi broj:

```
   11000000 10101000 00000101 10001100
 & 11111111 11111111 11111111 10000000
  ----------------------------------
   11000000 10101000 00000101 10000000
   11000000 10101000 00000101 10000000
```

→jednaki su

U C-u: `if ((adr2 & maska) == podmreza) {...`

Rad s operatorima pomaka bitova

- Operatori **<<** i **>>** služe za pomak svih bitova vrijednosti lijevog operanda u lijevo ili u desno.
- Broj pomaka u lijevo ili desno određen je desnim operandom.
- Pomak bitova za jedno mjesto u lijevo odgovara množenju vrijednosti operanda s 2, dok pomak za jedno mjesto u desno rezultira dijeljenjem vrijednosti operanda s 2.

Primjeri s pomakom bitova

- Primjer: Izračunati izraz **`2 << 1`**

  `0000 0000 0000 0000 0000 0000 0000 0010 (2)`

- Nakon pomaka u lijevo za jedno mjesto, rezultat je umnožak vrijednosti s 2:

  `0000 0000 0000 0000 0000 0000 0000 0100 (4)`

- Primjer: Izračunati izraz **`37 >> 2`**

  `0000 0000 0000 0000 0000 0000 0010 0101 (37)`

- Nakon pomaka u desno za dva mjesta, rezultat je cjelobrojno dijeljenje s 4:

  `0000 0000 0000 0000 0000 0000 0000 1001 (9)`

Primjeri s pomakom bitova

- Oprez: voditi računa o rasponu brojeva koji se mogu prikazati
- Primjer:

```
unsigned int i = 0x80000000, j;
j = i << 1;
printf("i=%u\nj=%u", i, j);
```

```
i=2147483648
j=0
```

- Primjer:

```
signed int i = 0x40000000, j;
j = i << 1;
printf("i=%d\nj=%d", i, j);
```

```
i=1073741824
j=-2147483648
```

Prioritet operatora

	OPERATORI	PRIDRUŽIVANJE
↑ Viši prioritet	poziv funkcije `()` `[]` referenciranje `.` postfiks `++ --`	L → D
	`! ~ ++ -- sizeof & *` prefiks `++ --` unarni `+ -`	D → L
	`(cast)`	D → L
	`* / %`	L → D
	binarni `+ -`	L → D
	`<< >>`	L → D
	`< <= > >=`	L → D
	`== !=`	L → D
	`&`	L → D
	`^`	L → D
	`|`	L → D
	`&&`	L → D
	`||`	L → D
	`? :`	D → L
	`= *= /= %= += -= &= ^= |= <<= >>=`	D → L
Niži prioritet ↓	`,`	L → D

Složeniji primjer s logičkim operatorima

- **Primjer:** Što će se ispisati sljedećim programskim odsječkom?

```
int a, b, c, d;
a = 0;
b = 4;
c = (a++) + b;
printf("a = %d, b = %d, c = %d ", a, b, c);
```

0 + 4 = 4, **a** se uvećava naknadno

```
a = 1, b = 4, c = 4
```

```
d = c && b + 3 * a;
printf("d = %d", d);
```

```
d = c && b + 3 * a
d = c && (b + (3 * a))
d = 4 && (4+ (3 * 1))
d = 4 && 7
d = 1
```

Ternarni operatori
Operator za uvjetno pridruživanje

- Operator za uvjetno pridruživanje (? :) je ternarni operator (zahtijeva tri operanda), a koristi se u pojedinim situacijama umjesto if-else naredbe. Oblik izraza u kojem se koristi operator je:

```
uvjetni_izraz ? izraz1 : izraz2;
```

- uvjetni_izraz se izračunava prvi. Ako je rezultat *true*, izračunava se **`izraz1`** i to je konačni rezultat cijelog izraza. Ako je uvjetni_izraz *false*, izračunava se **`izraz2`** i to je konačni rezultat cijelog izraza.

Operator za uvjetno pridruživanje

- Primjer: ako varijabla **c** sadrži znak koji predstavlja znamenku, u znakovnu varijablu **r** pohraniti vrijednost 'Z', a inače pohraniti vrijednost 'N'.
- Rješenje s pomoću *if-else*

```
if (c >= '0' && c <= '9') {
   r = 'Z';
} else {
   r = 'N';
}
```

- Rješenje s pomoću ternarnog operatora

```
r = c >= '0' && c <= '9' ? 'Z' : 'N';
```

Primjeri

- Učitati dva cijela broja. Ako je drugi broj različit od nule ispitati je li prvi broj djeljiv s drugim te ispisati jednu od sljedećih poruka:

  Broj *xxx* jest djeljiv s brojem *xxx*.

  Broj *xxx* nije djeljiv s brojem *xxx*.

```
#include <stdio.h>
int main(void) {
  int a, b;
  printf("Unesite a i b:");
  scanf("%d %d", &a, &b);
  if (b != 0) {
    printf("Broj %d %s djeljiv s brojem %d.\n",
           a,
           a % b ? "nije" : "jest",
           b);
  }
  return 0;
}
```

Primjeri

- Što će se ispisati sljedećim programskim odsječkom?

```
int a = 5, b = 10, c;
c = 1 ? ++a : ++b;
printf("a = %d, b = %d, c = %d \n", a, b, c);
```

```
a = 6, b = 10, c = 6
```

- Što će se ispisati sljedećim programskim odsječkom?

```
int a = 5, b = 10, c;
(c = 1) ? ++a : ++b;
printf("a = %d, b = %d, c = %d \n", a, b, c);
```

```
a = 6, b = 10, c = 1
```

Primjeri

- Koja je vrijednost varijable d nakon sljedećeg programskog odsječka:

```
short int a=4, b=2, c=8, d;
d = a < 10 && 2 * b < c;
```

- Rješenje:

```
d = ( a < 10 ) && ( ( 2 * b ) < c )
d = 1 && (4<8)
d = 1 && 1
d = 1
```

- Što će se ispisati sljedećim programskim odsječkom?

```
int a = 5, b = -1, c = 0;
c = (a = c && b) ? a = b : ++c;
printf("a = %d, b = %d, c = %d: \n", a, b, c);
```

- Rezultat:

```
a = 0 , b = -1 , c = 1
```

Skraćeni izrazi pridruživanja

- U programiranju se često nova vrijednost varijable izračunava na temelju stare vrijednosti te iste varijable. Zbog toga u C-u postoje **skraćeni izrazi pridruživanja**.
- **Primjer**

Izraz

```
i = i + 5;
```

može se pisati kao:

```
i += 5;
```

Izraz

```
i = i / (a + b);
```

može se pisati kao:

```
i /= a + b;
```

Skraćeni izrazi pridruživanja

- Gdje su skraćeni izrazi pridruživanja korisni?

  Članu niza **`niz`** s indeksom **`3*i+2*j-m*k`** uvećaj vrijednost za 9.

```
niz[3*i+2*j-m*k] = niz[3*i+2*j-m*k] + 9;
```

ili

```
niz[3*i+2*j-m*k] += 9;
```

Skraćeni izrazi pridruživanja

```
i = i + 7;              ⇨    i += 7;
j = j - k;              ⇨    j -= k;
a = a * (3 + 2);        ⇨    a *= 3 + 2;
b = b / (c * 2);        ⇨    b /= c * 2;
d = d % 2;              ⇨    d %= 2;
a = a & b;              ⇨    a &= b;
a = a ^ b;              ⇨    a ^= b;
a = a | b;              ⇨    a |= b;
a = a << b;             ⇨    a <<= b;
a = a >> b;             ⇨    a >>= b;
```

Primjer: što se obavlja sljedećim programskim odsječkom

```
int i, j, k;
...
k = i > j ? ++i : ++j;
```

Ako je **`i > j`**, **`i`** se uvećava za 1, sadržaj uvećanog **`i`** se pridružuje varijabli **`k`**, sadržaj varijable **`j`** se ne mijenja. U suprotnom, **`j`** se uvećava za 1, sadržaj uvećanog **`j`** se pridružuje varijabli **`k`**, sadržaj varijable **`i`** se ne mijenja.

Višestruko pridruživanje

- Operator pridruživanja obavlja pridruživanje vrijednosti izraza s desne strane operandu s lijeve strane (*L-value*). *L-value* mora imati dobro definiranu adresu i nakon izračunavanja izraza.
- Primjer:

```
a = b = c = 0;
```

- Sve tri varijable inicijaliziraju se na vrijednost 0. Pridruživanje je s desna na lijevo, tj. kao da je napisano:

```
a = (b = (c = 0));
```

- Prvo se varijabli c pridružuje 0 i vrijednost tog cijelog izraza (c = 0) poprima vrijednost 0; zatim se varijabli b pridružuje vrijednost tog izraza, zatim cijeli taj izraz poprima vrijednost 0 koja se na kraju pridružuje varijabli a.

```
a = b = c + 3;
```

- Može li?

```
a = b + 3 = c = d * 3;
```

NE MOŽE!!! → b+3 nije l-value

Odvajanje naredbi zarezom

- Zarez se kao operator koristi za odvajanje naredbi obično tamo gdje je dopuštena samo jedna naredba. Na primjer:

```
float x, y;
x = 7.0f, y = 3.0f;
```

Poteškoće sa složenim logičkim uvjetima

- Npr. izraz "ako je x veći od 20 i manji od 100", pogrešno je napisati tako da se umjesto operatora "I" stavi operator ","

```
if ( x > 20, x < 100)
```

što dovodi do logičke pogreške koju je teško otkriti.

Izraz

```
x > 20, x < 100
```

je pravopisno ispravan, ali po rezultatu odgovara izrazu

```
x < 100
```