Programiranje 2

Milena Vujošević Janičić
Jelena Graovac

www.matf.bg.ac.rs/~milena
www.matf.bg.ac.rs/~jgraovac

Beograd, 13. februar, 2020.

Pregled

Pregled

Operatori za rad nad bitovima

- C podržava naredne operatore za rad nad pojedinačnim bitovima, koji se mogu primenjivati samo na celobrojne argumente
  - ~ bitovska negacija;
  - & bitovska konjunkcija;
  - | bitovska disjunkcija;
  - ^ bitovska ekskluzivna disjunkcija;
  - << pomeranje (šiftovanje) bitova ulevo;
  - >> pomeranje (šiftovanje) bitova udesno;
  - &=, |=, ^=, >>=, >>= – bitovske dodele — ovi operatori kombinuju bitovske operatore sa dodelom

Operatori za rad nad bitovima

- C podržava naredne operatore za rad nad pojedinačnim bitovima, koji se mogu primenjivati samo na celobrojne argumente
  - `~` bitovska negacija;
  - `&` bitovska konjunkcija;
  - `|` bitovska disjunkcija;
  - `^` bitovska ekskluzivna disjunkcija;
  - `<<` pomeranje (šiftovanje) bitova ulevo;
  - `>>` pomeranje (šiftovanje) bitova udesno;
  - `&=`, `|=`, `^=`, `>>=`, `>>=` – bitovske dodele — ovi operatori kombinuju bitovske operatore sa dodelom

Prioritet operatora i asocijativnost

1. (), [ ]	sleva nadesno
2. !, ~, ++,--, +, -, *, &, (tip)	zdesna nalevo
3. *, /, %	sleva nadesno
4. +, -	sleva nadesno
5. <<, >>	sleva nadesno
6. <, <=, >, >=	sleva nadesno
7. ==, !=	sleva nadesno
8. &	sleva nadesno
9. ^	sleva nadesno
10. |	sleva nadesno
11. &&	sleva nadesno
12. ||	sleva nadesno
13. ?:	zdesna ulevo
14. =,+=,-=,/=,%=,&=,^=,|=,<<=,>>=	zdesna ulevo
15. ,	sleva nadesno

Prioritet operatora

- Kao unarni operator, operator ~ ima najveći prioritet i desno je asocijativan.
- Prioritet operatora pomeranja je najveći od svih binarnih bitovskih operatora — nalazi se između prioriteta aritmetičkih i relacijskih operatora.
- Ostali bitovski operatori imaju prioritet između relacijskih i logičkih operatora
- & ima veći prioritet od ^ koji ima veći prioritet od |. Ovi operatori imaju levu asocijativnost.
- Operatori dodele imaju skoro najniži prioritet i desnu asocijativnost

Bitovski operatori

A	B	A ∧ B
0	0	
0	1	
1	0	
1	1	

A	B	A ∨ B
0	0	
0	1	
1	0	
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	¬ A
0	
1	

Bitovski operatori

A	B	A ∧ B
0	0	0
0	1	
1	0	
1	1	

A	B	A ∨ B
0	0	
0	1	
1	0	
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	¬ A
0	
1	

Bitovski operatori

A	B	A ∧ B
0	0	0
0	1	0
1	0	
1	1	

A	B	A ∨ B
0	0	
0	1	
1	0	
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	¬ A
0	
1	

Bitovski operatori

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	

A	B	A $\vee$ B
0	0	
0	1	
1	0	
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	$\neg$ A
0	
1	

Bitovski operatori

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A $\vee$ B
0	0	
0	1	
1	0	
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	$\neg$ A
0	
1	

Bitovski operatori

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A ∨ B
0	0	0
0	1	
1	0	
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	¬ A
0	
1	

Bitovski operatori

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A $\vee$ B
0	0	0
0	1	1
1	0	
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	$\neg$ A
0	
1	

Bitovski operatori

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A ∨ B
0	0	0
0	1	1
1	0	1
1	1	

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	¬ A
0	
1	

Bitovski operatori

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A $\vee$ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A $\underline{\vee}$ B
0	0	
0	1	
1	0	
1	1	

A	$\neg$ A
0	
1	

Bitovski operatori

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A ∨ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A $\underline{\vee}$ B
0	0	0
0	1	
1	0	
1	1	

A	¬ A
0	
1	

Bitovski operatori

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A ∨ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A $\underline{\vee}$ B
0	0	0
0	1	1
1	0	
1	1	

A	¬ A
0	
1	

Bitovski operatori

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A $\vee$ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A $\underline{\vee}$ B
0	0	0
0	1	1
1	0	1
1	1	

A	$\neg$ A
0	
1	

Bitovski operatori

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A ∨ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A $\underline{\vee}$ B
0	0	0
0	1	1
1	0	1
1	1	0

A	¬ A
0	
1	

Bitovski operatori

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A $\vee$ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A $\underline{\vee}$ B
0	0	0
0	1	1
1	0	1
1	1	0

A	$\neg$ A
0	1
1	

Bitovski operatori

A	B	A $\wedge$ B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A $\vee$ B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A $\underline{\vee}$ B
0	0	0
0	1	1
1	0	1
1	1	0

A	$\neg$ A
0	1
1	0

Bitovsko i

- `&` – bitovsko `i` — primenom ovog operatora vrši se konjunkcija pojedinačnih bitova dva argumenta — i-ti bit rezultata predstavlja konjunkciju i-tih bitova argumenata.

Primer

```
  207 & 63 = 15

  00000000000000000000000011001111  <---207
& 00000000000000000000000000111111  <---63
================================
  00000000000000000000000000001111  <---15
```

Bitovsko ili

- | – bitovsko `ili` — primenom ovog operatora vrši se disjunkcija pojedinačnih bitova dva argumenta — i-ti bit rezultata predstavlja disjunkciju *i*-tih bitova argumenata.

Primer

```
  207 | 63 = 255

  00000000000000000000000011001111  <---207
| 00000000000000000000000000111111  <---63
==================================
  00000000000000000000000011111111  <---255
```

Bitovska negacija

- ~ – jedinični komplement — primenom ovog operatora vrši se komplementiranje pojedinačnih bitova argumenta — i-ti bit rezultata predstavlja komplement *i*-tog bita argumenta

Primer

```
~63 = -64 (jer je u pitanju celobrojna oznacena
           konstanta)

00000000000000000000000000111111 <--- 63
================================
11111111111111111111111111000000 <--- -64
```

Bitovsko xor

- ^ – bitovsko xor — primenom ovog operatora vrši se ekskluzivna disjunkcija pojedinačnih bitova dva argumenta — i-ti bit rezultata predstavlja ekskluzivna disjunkcija i-tih bitova argumenata.

Primer

```
  207 ^ 63 = 240

  00000000000000000000000011001111  <---207
^ 00000000000000000000000000111111  <---63
====================================
  00000000000000000000000011110000  <---240
```

Šiftovanje u levo

- << – levo pomeranje (šiftovanje) — primenom ovog operatora bitovi prvog argumenta se pomeraju u levo za broj pozicija naveden kao drugi argument. Početni bitovi prvog argumenta se zanemaruju, dok se na završna mesta rezultata upisuju nule.

Primer

```
63 << 3 = 504

00000000000000000000000000111111  <--- 63
================================ << 3
00000000000000000000000111111000  <--- 504
```

- Levo pomeranje za jednu poziciju odgovara množenju sa dva.
- Levo pomeranje za n pozicija odgovara množenju sa n-tim stepenom dvojke.

Šiftovanje u desno

- >> — desno pomeranje (šiftovanje) — primenom ovog operatora bitovi prvog argumenta se pomeraju u desno za broj pozicija naveden kao drugi argument.
- Krajnji (desni) bitovi prvog argumenta se zanemaruju
- Početni (levi) bitovi prvog argumenta:
  - logičko pomeranje — popunjavaju se nulama
  - aritmetičko pomeranje — popunjavanje bitovima znaka
- Aritmetičko pomeranje za jedno mesto — deljenje sa dva

Šiftovanje u desno

- Koje pomeranje će se vršiti zavisi od tipa prvog argumenta:
  - neoznačen tip — logičko pomeranje
  - označen tip — aritmetičko pomeranje

Primer

```
63 >> 3 = 7

00000000000000000000000000111111  <--- 63
================================ >> 3
00000000000000000000000000000111  <--- 7
```

Napomena

- Bitovske operatore ne treba mešati sa logičkim operatorima.
- Na primer, vrednost izraza `1 && 2` je 1 (tačno i tačno je tačno) dok je vrednost izraza `1&2` jednaka 0

```
  1 & 2 = 0

  00000000000000000000000000000001   <--- 1
& 00000000000000000000000000000010   <--- 2
  ================================
  00000000000000000000000000000000   <--- 0
```

Primena bitovskih operatora

- Kako bi se postigao željeni efekat nad bitovima nekog broja, običaj je da se vrši njegovo kombinovanje bitovskim operatorima sa specijalno pripremljenim konstantama koje se nazivaju maske.
- Za građenje maski koriste se osobine bitovskih operatora

Primena bitovskog i

- Osobina konjunkcije je da je za svaki bit `b`, vrednost `b & 0` jednaka 0, dok je vrednost `b & 1` jednaka `b`.
- Ovo znači da se konjunkcijom sa nekom maskom dobija rezultat koji je jednak broju koji se dobije kada se u broj x upišu nule na sve one pozicije na kojima maska ima bit 0, dok ostali bitovi ostaju neizmenjeni.

Primena bitovskog i

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 0, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko & sa maskom koja se sastoji od svih jedinica, a jedino na *i*-tom mestu je bit nula.

Primena bitovskog i

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 0, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko & sa maskom koja se sastoji od svih jedinica, a jedino na *i*-tom mestu je bit nula. Ova maska se dobija negacijom maske koja sadrži sve nule, a na i-tom mestu jedinicu.

Primena bitovskog i

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 0, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko & sa maskom koja se sastoji od svih jedinica, a jedino na *i*-tom mestu je bit nula. Ova maska se dobija negacijom maske koja sadrži sve nule, a na i-tom mestu jedinicu.
Maska koja sadrži na *i*-tom mestu jedinicu može se dobiti pomeranjem broja 1 za *i* mesta u levo.

Kôd:

Primena bitovskog i

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 0, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko & sa maskom koja se sastoji od svih jedinica, a jedino na *i*-tom mestu je bit nula. Ova maska se dobija negacijom maske koja sadrži sve nule, a na i-tom mestu jedinicu. Maska koja sadrži na *i*-tom mestu jedinicu može se dobiti pomeranjem broja 1 za *i* mesta u levo.

Kôd:

```
n & (~(1 << i))
```

Primena bitovskog i

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 0, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko `&` sa maskom koja se sastoji od svih jedinica, a jedino na *i*-tom mestu je bit nula. Ova maska se dobija negacijom maske koja sadrži sve nule, a na i-tom mestu jedinicu. Maska koja sadrži na *i*-tom mestu jedinicu može se dobiti pomeranjem broja 1 za *i* mesta u levo.

Kôd:

```
n & (~(1 << i))
```

Primena bitovskog ili

- Osobina disjunkcije je da je za svaki bit b, vrednost `b | 0` jednaka b, dok je vrednost `b | 1` jednaka 1.
- Dakle, disjunkcijom broja x sa nekom maskom dobija se rezultat u kojem su upisane jedinice na sve one pozicije na kojima maska ima bit 1, dok ostali bitovi ostaju neizmenjeni.

Primena bitovskog ili

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 1, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko | sa maskom koja se sastoji od svih nula, a jedino na *i*-tom mestu je bit 1.

Primena bitovskog ili

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 1, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko | sa maskom koja se sastoji od svih nula, a jedino na *i*-tom mestu je bit 1. Maska koja sadrži na *i*-tom mestu jedinicu može se dobiti pomeranjem broja 1 za *i* mesta u levo.

Kôd:

Primena bitovskog ili

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 1, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko | sa maskom koja se sastoji od svih nula, a jedino na *i*-tom mestu je bit 1. Maska koja sadrži na *i*-tom mestu jedinicu može se dobiti pomeranjem broja 1 za *i* mesta u levo.

Kôd:

```
n | (1 << i)
```

Primena bitovskog ili

Primer

Napisati naredbu koja na *i*-to mesto broja *n* postavlja bit 1, dok ostali bitovi ostaju nepromenjeni.

Rešenje:
Potrebno je uraditi bitovsko | sa maskom koja se sastoji od svih nula, a jedino na *i*-tom mestu je bit 1. Maska koja sadrži na *i*-tom mestu jedinicu može se dobiti pomeranjem broja 1 za *i* mesta u levo.

Kôd:

```
n | (1 << i)
```

Primena bitovskog xor

- Osobina ekskluzivne disjunkcije je da za svaki bit `b`, vrednost `b ^ 0` jednaka `b`, dok je vrednost `b ^ 1` jednaka `~b`.
- Ovo znači da se ekskluzivnom disjunkcijom sa nekom maskom dobija rezultat koji je jednak broju koji se dobije kada se invertuju bitovi na onim pozicijama na kojima se u maski nalazi jedinica.

Primena bitovskog xor

Primer

Napisati naredbu koja invertuje bit na i-tom mestu broja n.

Rešenje:
Potrebno je uraditi bitovsko ^ sa maskom koja se sastoji od svih nula, a jedino na i-tom mestu je bit 1. Maska koja sadrži na i-tom mestu jedinicu može se dobiti pomeranjem broja 1 za i mesta u levo.

Primena bitovskog xor

Primer

Napisati naredbu koja invertuje bit na i-tom mestu broja n.

Rešenje:
Potrebno je uraditi bitovsko ^ sa maskom koja se sastoji od svih nula, a jedino na i-tom mestu je bit 1. Maska koja sadrži na i-tom mestu jedinicu može se dobiti pomeranjem broja 1 za i mesta u levo.

Kôd:

```
n ^ (1 << i)
```

Primena bitovskog xor

Primer

Napisati naredbu koja invertuje bit na i-tom mestu broja n.

Rešenje:
Potrebno je uraditi bitovsko ^ sa maskom koja se sastoji od svih nula, a jedino na i-tom mestu je bit 1. Maska koja sadrži na i-tom mestu jedinicu može se dobiti pomeranjem broja 1 za i mesta u levo.

Kôd:

```
n ^ (1 << i)
```

Bitovski operatori - osnovna pravila

- Ukoliko želimo da u okviru broja postavimo nule na neke izabrane pozicije, radimo bitovsko i sa maskom koja sadrži nule na mestima gde želimo da postavimo nulu, a na svim ostalim mestima maska treba da sadrži jedinice
- Ukoliko želimo da u okviru broja postavimo jedinice na neke izabrane pozicije, radimo bitovsko ili sa maskom koja sadrži jedinice na mestima gde želimo da postavimo jedinice, a na svim ostalim mestima maska treba da sadrži nule
- Ukoliko želimo da invertujemo bitove na određenim pozicijama, radimo bitovsku ekskluzivnu disjunkciju sa maskom koja sadrži jedinice na mestu gde želimo da radimo invertovanje, a na svim ostalim mestima maska treba da sadrži nule

Pregled

Literatura

- Slajdovi su pripremljeni na osnovu knjiga:
  Filip Marić, Predrag Janičić: Programiranje 1
  Predrag Janičić, Filip Marić: Programiranje 2
- Za pripremu ispita, slajdovi nisu dovoljni, neophodno je učiti iz knjige!