Programski jezici i strukture podataka

7

Rad sa binarnim datotekama - čitanje

```
int fread (void *niz, int vel, int br, FILE *dat) ;
```

- Ova funkcija čita iz datoteke dat najviše **br** podataka veličine **vel bajtova** u memoriju počev od adrese **niz**. Čitanje podataka počinje na poziciji gde je prethodni pristup datoteci završen.
- Po završetku čitanja zaustavlja se neposredno iza poslednjeg pročitanog bajta.

Rad sa binarnim datotekama - čitanje

- Dužnost je programera da vodi računa o logičkoj strukturi datoteke, tj. o veličini pojedinih podataka, njihovom broju i redosledu.
- U slučaju neusaglašenosti čitanja prema ranijim upisivanjima, dobiće se besmisleni podaci bez ikakve opomene.

Rad sa binarnim datotekama - čitanje

- Vrednost funkcije je stvarni broj pročitanih podataka. Jedan karakterističan siučaj kada je vrednost funkcije manja od br je kada od trenutne pozicije u datoteci do kraja datoteke nema dovoljan broj podataka.
- Primer kojim se iz datoteke podaci vrši čitanje niza od najviše n_max celobrojnih podatka:

```
int n = fread (vektor, sizeof (int), n_max, podaci);
```

Rad sa binarnim datotekama - pisanje

```
fwrite (const void *niz, int vel, int br, FILE *dat);
```

- Ova funkcija piše **br** podataka veličine **vel** bajtova iz memorije počev od adrese niz u datoteku dat.
- Pisanje podataka počinje na poziciji gde je prethodni pristup datoteci završen.
- Ako trenutna pozicija nije na kraju datoteke, nove vrednosti se prepisuju preko zatečenih vrednosti u datoteci.

Rad sa binarnim datotekama - pisanje

- Ako do kraja datoteke nema dovoljno mesta za smeštanje svih podataka, pisanje se nastavlja iza kraja datoteke.
- Time se povećava veličina datoteke.
- Po završetku pisanja zaustavlja se neposredno iza poslednjeg upisanog bajta.

Rad sa binarnim datotekama - pisanje

- Dužnost je programera da vodi računa o logičkoj strukturi datoteke, tj. o veličini pojedinih podataka, njihovom broju i redosledu.
- U slučaju neusaglašenosti prilikom prepisivanja preko starog sadržaja prema ranijim upisivanjima, oštetiće se logička struktura datoteke bez ikakve opomene.

Rad sa binarnim datotekama - pisanje

- Vrednost funkcije je broj prenetih podataka.
- Ta vrednost je u slučaju greške manja od br.
- Primer kojim se u datoteku podaci piše niz od n celobrojnih podatka:

```
greska = fwrite (vektor, sizeof (int), n, podaci) < n;
```

Pozicioniranje unutar datoteke (direktan pristup)

```
int fseek (FILE *dat, long pomeraj, int reper) ;
```

- Ova funkcija vrši pozicioniranje na mesto u datoteci dat čija je udaljenost pomeraj bajtova od označene reperne tačke.
- Moguće reperne tačke obeležavaju se simboličkim konstantama **SEEK_SET** (početak datoteke), **SEEK_CUR** (trenutna pozicija u datoteci) ili **SEEK_END** (kraj datoteke).
- Sledeće čitanje ili pisanje vršiće se počevši od ovako odabrane pozicije u datoteci.

UNIJE

Unija

- Unija je struktura koja može čuvati (u različito vreme) objekte različitih tipova i veličina. Time se obezbeđuje manipulisanje različitim vrstama podataka u istom memorijskom području.
- Unija za razliku od stukture zauzima samo onoliko prostora koliko je dovoljno za smeštanje njenog najvećeg člana. Zbog toga je u svakom trenutku moguće koristiti samo jedan od članova unije.

- Sintaksa unije analogna je sintaksi strukture:

```
union  ime
{
    tip_1 ime_1;
    tip_2 ime_2;
    ....  ....
    tip_n ime_n;
};
```

- Varijabla ime može se deklarisati kao:

```
union  ime  x,y;
```

POLJA BITOVA

Podela memorije

- Pomocu polja bitova, postiže se kompaktnost memorije u C jeziku.
- Zona memorije podeli se na skup manjih zona koje imaju ograničen opseg vrednosti.

- Za brojač od 0 do 7, potrebna su samo tri bita.

Definicija polja bitova

```
struct tacka_st {
  unsigned int red : 5;
  unsigned int kol : 7;
  unsigned int vidljiv : 1;
  unsigned int tabelaboja : 3;
};
```

- Ovom definicijom kreira se šablon sa kojim se mogu deklarisati promenljive.

Deklarisanje promenljive tipa polje bitova

- Polja bitova se deklarišu samo kao označeni ili neoznačeni celobrojni tipovi.
- Za svaku promenljivu ili element niza rezerviše se navedeni broj bitova.
- Deklarisanje promenljive tipa polje bitova - specifikator tipa i ime:

```
struct tacka_st tacka;
```

Pristup vrednosti

- Posle deklarisanja promenljive tacka, možete da pristupite poljima bitova koristeci istu notaciju kao za pristup članovima strukture.

```
tacka.red = 12;
```

- Može se deklarisati pokazivač na promenljivu tipa polje bitova i koristiti notacija ->:

- Pojedinačna polja bitova nemaju adrese!

BITWISE

Operacije nad bitovima

Programski jezik C ima jedan broj operatora čije je delovanje definisano na bitovima.
Takvi se operatori mogu primeniti na celobrojne tipove podataka `char, short, unsigned, int i long`.

To su sledeći operatori:

Operator	Značenje
&	logičko I bit-po-bit
|	logičko ILI bit-po-bit
^	ekskluzivno logičko ILI bit-po-bit
<<	levi pomak (left-shift)
>>	desni pomak (right-shift)
~	1-komplement

`&= |= ^= <<= >>=`

Operacije nad dva operanda

Prve tri operacije &, | i ^ uzimaju dva operanda i vrše operacije na bitovima koji se nalaze na odgovarajućim mestima. Upoređuju se bitovi na najmanje značajnom mestu u oba operanda, zatim na sledećem najmanje značajnom mestu itd.

Definicije operacija date su u sledećoj tabeli:

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

Maske (bit masking)

Logički operatori najčešće služe maskiranju pojedinih bitova u operandu. Maskiranje je transformacija binarnog zapisa u varijabli na određen način.

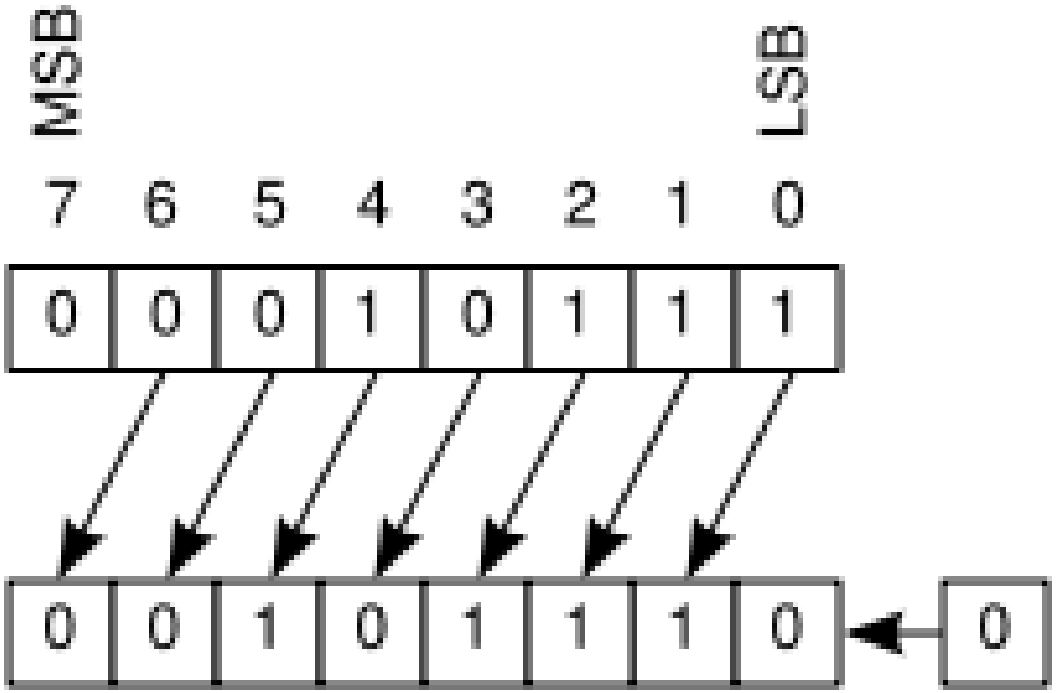
Na primer, pretpostavimo da želimo šest najmanje značajnih bitova iz varijable `a` kopirati u varijablu `b`, a sve ostale bitove varijable `b` staviti na nulu.
To možemo postići pomoću logičkog I operatora. Pretpostavimo da su Varijable tipa int i da je int kodiran u 16 bitova. Tada prvo definišemo masku, konstantu koja ima sledeći raspored bitova:

```
mask = 0000 0000 0011 1111 (ili 0x3f)
```

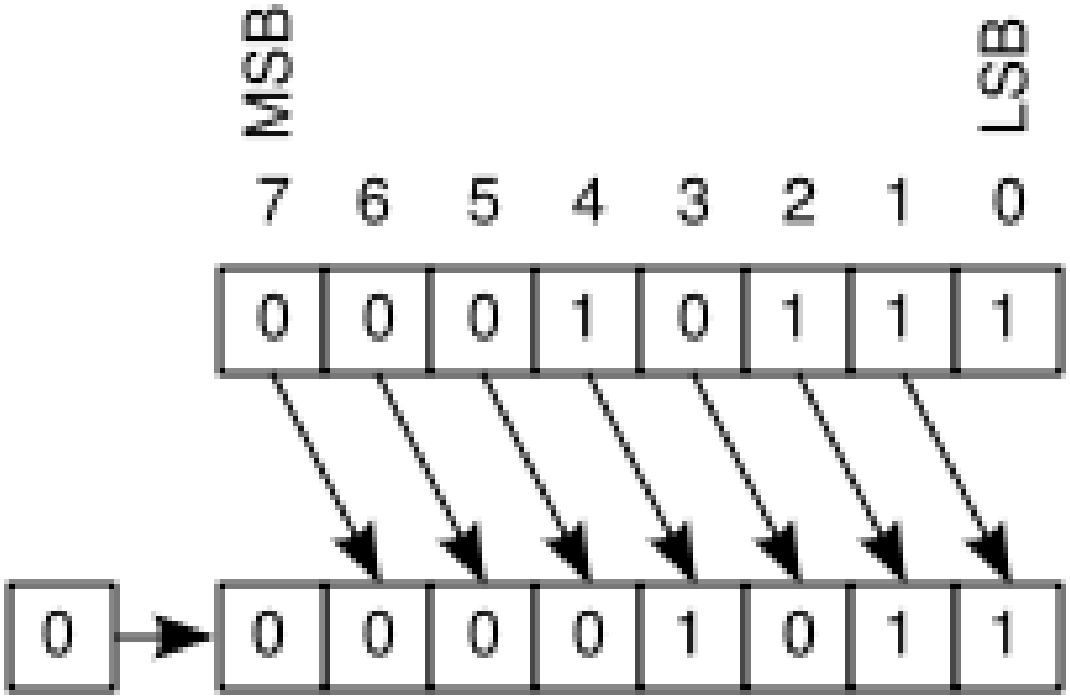
Operacija koju trebamo napraviti je

```
      a = 0100 0111 0101 0111
   mask = 0000 0000 0011 1111
          ---------------------
a & mask =0000 0000 0001 0111
```

left-shift



right-shift



Primer 1 - Za uneseni pozitivni broj ispisuje se koliko on ima jedinica u binarnom zapisu.**z35.c**

Primer 2 - Ispisivanje u binarnom brojnom sistemu.

z36.c

DINAMIČKA ALOKACIJA MEMORIJE

Upravljanje memorijom - potreba

- Često ne znate obim podataka s kojima će program raditi ili to ne možete precizno da procenite.
- Potrebno je memoriju zauzimati tokom izvršavanja samo u potrebnoj meri i što pre je osloboditi.
- Ne menjati program da bi radio s većom količinom podataka .

Dinamičko upravljanje memorijom

- Zahtevi iz procesa za dodatnim memorijskim resursima.
- To zavisi od količine slobodne memorije u području obično zvanom "heap"
- Heap predstavlja područje nezauzete memorije koja se na zahtev dodeljuje procesu.

Podrška u C jeziku za dinamičku alokaciju memorije

- Standardna biblioteka sadrži četiri funkcije za dinamičko upravljanje memorijom:

Dodeljuju nov blok u memoriji

- `malloc()`
- `calloc()`

Menja veličinu dodeljenog bloka

- `realloc()`

Oslobađa dodeljenu memoriju

- `free()`

Sve četiri funkcije deklarisane su u datoteci zaglavlja `stdlib.h`.

malloc

```
void *malloc (size_t size);
```

- Za argument funkcije uzima veličinu bloka koji želimo alocirati
- Vraća generički pokazivač (pokazivač bez tipa) na početak bloka (tj, vraća adresu prvog bajta bloka).
- Ako alokacija nije uspešna, vraća **NULL**.
- Za upotrebu malloc trebamo uključiti,

```
#include<stdlib.h>
```

(z37.c)

calloc

```
void *calloc(n, size);
```

- Rezerviše memorijski blok dovoljan za memorisanje n elemenata svaki veličine size bajtova, znači n*size.
- Rezervisan memorijski blok je inicijalizovan na 0.
- U slučaju uspešne rezervacije calloc vraća generički pokazivač (pokazivač bez tipa) koji pokazuje na rezervisan memorijski blok. U protivnom, vraća **NULL**.

realloc

```
void *realloc(pokaz, size);
```

- Oslobađa rezervisani memorijski blok i rezerviše novi veličine size bajtova.
- Argument **`pokaz`** pokazuje na memorijski blok, koji se realocira.
- Argument **`size`** je unsigned i određuje veličinu realociranog memorijskog bloka.
- Ako je realociranje uspoešno **`realloc`** vraća generički pokazivač koji pokazuje na movi memorijski blok. U protivnom, vraća NULL.

free

- Kada nam memorijski blok koji alcoran s malloc nije više potreban, moramo ga osloboditi. To se radi s funkcijom

`free(arg)`

- Argument funkcije **`free()`** mora biti pokazivač koji pokazuje na početak prostora koji želimo osloboditi (dealocirati).
- ”curenje memorije”, rezervisanje bez oslobađanja
- Za svaki malloc() moramo imati jedan **`free()`** !