Programski jezici i strukture podataka

9

STRUKTURE

Strukture u C-u

- Jezik C ima skup osnovnih tipova podataka i od njih se na nekoliko načina dobijaju tipovi višeg nivoa.
- Podatke koji imaju neku međusobnu vezu možete da grupišete pod isto ime i da im pristupite pomoću tog imena i izbora unutrašnjeg člana.

Deklaracije struktura

- Opšti oblik deklaracije strukture u jeziku C je sledeći:

```
struct oznaka {
                tip imeprom;
                };
```

- Deklaracija strukture započinje ključnom reči ***struct***.
- Opciona ***oznaka*** daje konkretnoj strukturi jedinstveno ime.
- Lista deklaracija promenljivih specificira članove strukture.
- Proizvod ovakve deklaracije je šablon strukture koji možete da koristite kao specifikator tipa.

```
struct oznaka{
                tip imeprom;
              };
```

Šablon meseci u godini

- Evo deklaracije koja kreira šablon za strukturu korišcenu za smeštaj informacije o mesecima u godini:

```
struct mesec_st { char ime[10];
                  char skr[4];
                  short dana;
                     };
```

- Sufiksi "_st" ili "_t" su uobičajeni za konvenciju da je ime koje ga sadrži oznaka strukture.
- U standardnoj biblioteci jezika C mnogi izvedeni tipovi imaju sufiks "_t" .

Deklaracija promenljive

- Dva niza znakova u ovoj strukturi zauzimaju prostor fiksne veličine u memoriji.
- U članu ***ime*** se sadrži ime meseca plus završni nulti znak.
- U članu ***skr*** je skracenica od tri slova plus njen završni nulti znak.
- Član ***dana*** je kratak ceo broj, koji zauzima dva bajta.
- Pošto su članovi ime i skr nizovi, oni zauzimaju podjednako veliki memor- ijsko prostor bez obzira šta je u njih upisano.

```
struct mesec_st mesec;
```

Pristup elementima strukture

- Kada se deklariše strukturna promenljiva njenim članovima se mogu dodeliti ili u njih kopirati vrednosti.
- Da bi dodelili broj dana u mesecu januaru promenljivoj ***mesec***, koristi se:

```
mesec.dana = 31;
```

- Tačka se naziva operatorom člana strukture.
- Prvenstvo ovog operatora je na samom vrhu tabele prioriteta, zajedno sa funkcijom (zagrade), nizom (uglaste zagrade).
- Radi pristupa članu strukturne promenljive primenite operator tačka:

```
imeprom.clan
```

Strukture i funkcije

- Struktura može biti parametar funkcije.

```
int prestupGod(struct datum_st);
```

- Funkcija može da vrati strukturu onome ko ju je pozvao.

```
struct datum_st zadajDatum(char, char, char);
```

Pokazivači i strukture

- Izbor člana strukture pomoću pokazivača može se predstavit na dva načina:
  - standardnom notacijom, koristeci ime strukture i operator tačka;
  - korišćenjem pokazivača na strukturu ("->").

```
struct oznaka { tip m;...};

struct oznaka s, *p;//struktura i pokazivac
p = &s;//p sada pokazuje na s
```

- Bez korišcenja pokazivača, članu m pristupa se primenom operatora tačka:

```
s.m
```

- Kada se koristi prethodno inicijalizovan pokazivač, p, izraz *p je struktura (isto kao kada se s korist neposredno).

```
(*p).m
```

- Zagrade su ovde potrebne zato što operator tačka proizvodi čvršcu vezu nego operator posrednog pristupa (*).
- Izbor člana strukture preko pokazivača javlja se često u programima na jeziku C. Iz tog razloga postoji pogodniji oblik:

```
p->m
```

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.

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!

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()`** !

Dinamička alokacija memorije

- **Prilikom definisanja skalarnih promenljivih ili nizova, memorija se zauzima statički**
  - sve potiebe za memorijom moraju biti poznate unapred, pre prevođenja izvornog programa
  - alocira se najčešce više prostora nego što je potrebno
  - prostor se zauzima u statičkoj zoni memorije
  - bilo kakva izmena kapaciteta zahteva ponovno prevođenje programa
- **Rešenje - dinamička alokacija memorije**
  - radi se u vreme izvršavanja programa
  - broj i veličinu podataka ne moramo znati unapred
  - memorija se alocira u dinamičkoj zoni memorije (eng. heap)
  - podacima u dinamičkoj zoni memorije se pristupa preko pokazivača
  - ograničenje samo ukupan raspoloživ memorijski prostor u sistemu
- **Odgovornost programera za ispravnu alokaciju i dealokaciju dinamičke memorije!**

Manipulacija dinamičkom memorijom

- Zauzimanje dinamičke memorije

  void* **malloc**(vel)

  - **vel** predstavlja traženu veličinu memorije u bajtovima

  void* **calloc**(n,vel)

  - n predstavlja broj elemenata niza
  - **vel** predstavlja veličinu jednog elementa
  - inicijalizuje zauzeti prostor nulama
- Obe funkcije vraćaju generički (**void**) pokazivač!
  - ovakav pokazivač sadrži samo adresu, a nije poznat tip podatka na koji ukazuje
  - ne može se dereferencirati, niti koristiti u adresnoj aritmeticij, osim poređenja
  - ukoliko želimo da pristupimo memoriji preko njeqa, moramo izvršiti eksplicitnu konverziju (cast) u neki tip

STRUKTURE PODATAKA

- Linearne strukture podataka
  - Nizovi
  - Spregnute liste
  - Stekovi
  - Redovi
  - Dekovi
- Nelinearne strukture podataka
  - Stabla
  - Grafovi

Linearna struktura podataka

- Zajednički imenitelj za sve linearne strukture je postojanje serije podataka jednodimenzijalnog poretka.
- Za broj elemenata linearne strukture se kaže **dužina** i uobičajeno se obeležava sa n.
- Kada je n=0, linearna struktura je prazna.
- Kada je n>0, linearna struktura ima elemente.
- Prvi element **nema prethodnika**, poslednji element **nema sledbenika**, svi ostali elementi imaju i prethodnika i sledbenika.

Operacije nad linearnom strukturom podataka

- Pristup elementima strukture (0,...,n-1).
- Pretraživanje strukture i vraćanje pozicije identifikovanog elementa.
- Pristupanje vrednosti pojedinog elementa na pisanje ili čitanje.
- Umetanje novog elementa na proizvoljnu poziciju.
- Umetanje novog elementa pre prve pozicije.
- Umetanje novog elementa iza poslednje pozicije.
- Brisanje elementa.
- Brisanje cele strukture.
- Pronalaženje prethodnika ili sledbenika posmatranog elementa.
- Određivanje dužine strukture.
- Spajanje dve ili više struktura u jednu.
- Razdvajanje strukture na dve ili više.

Fizička realizacija koncepta linearne strukture podataka

- Sekvencijalna fizička realizacija.
- Spregnuta fizička realizacija.

- Strukture u obe fizičke realizacije mogu da podrže prethodno navedene operacije, ali sa različitom performansom.
  - Primer pristupa elementu...
  - Primer ubacivanja ili brisanja elementa...

Odabir fizičke realizacije zavisi od namene

- Za specifičnu implementaciju linearne strukture podataka, pre svega, treba detektovati namenu i podskup operacija koje treba realizovati.
- Pogotovo treba obratiti pažnju na izbor fizičke realizacije kod struktura sa specifičnim pravilima pristupa (FIFO, LIFO,...).