Programiranje I

Programske biblioteke

Karakteristike promjenljivih

Dinamička alokacija i dealokacija

Programske biblioteke

- Svi C kompajleri dolaze sa desetinama programskih biblioteka.
- Ovdje ćemo pobrojati neke od njih (najvažnije) i u njima samo najvažnije funkcije.
- Ostale ostavljamo za vježbu studentima, kao i budućim programerima za profesionalni rad.
- Podsjetimo se kratko biblioteke stdio.h iz koje smo naučili da koristimo: putchar(c), getchar(), gets(s) , puts(s), printf, scanf. Ujedno smo upoznali i konstantu NULL.

Još nešto o scanf

- Deklaracija funkcije scanf izgleda:

  int scanf(const char *format, ...)

  pri čemu format predstavlja string koji može sadržati:
  - karaktere bez bjelina
  - bjeline
  - specifikatore formata ulaznih podataka (%...)

- Rezultat scanf je cijeli broj koji predstavlja broj uspješno učitanih podataka. U suprotnom, funkcija vraća negativnu vrijednost ili EOF, u zavisnosti od greške koja se desila.

Specifikatori formata

% [*] [širina] [dužina] tip

- Zagrade [] označavaju da je polje opciono, tj. ne mora se navesti.
- * - ulazni podatak se učitava, ali se ignoriše (ne dodjeljuje se odgovarajućoj promjenljivoj)
- širina - maksimalan broj karaktera koji se učitava
  - %2d označava promjenljivu tipa int sa dvije cifre
  - %4f označava float promjenljivu sa ukupno 4 karaktera (počev od cifre najveće težine)
- dužina - dodatno određuje tip promjenljive u smislu veličine
  - %hd označava promjenljivu tipa short int
  - %ld označava promjenljivu tipa long int
- tip - karakter koji predstavlja tip podatka koji se učitava ili koji se očekuje na ulazu (npr. %d)

Specifikator opsega

- Specifikator za učitavanje određenog opsega karaktera sve dok se ne učita karakter koji ne pripada zadatom opsegu je:

  %[opseg] ⟵ Sad se ne navode spec. formata

  - %[0-9] znači - učitavaj sve dok su ulazni karakteri cifre od 0 do 9.
  - %[AG-M37] znači - učitavaj sve dok se ulazni podaci slovo A, neko od slova iz opsega od G do M, ili cifre 3 ili 7.
- Moguće je specificirati opseg karaktera koje je potrebno ignorisati pri učitavanju podataka, korišćenjem specifikatora:

  %[^opseg]

  - %[^A-C] - učitavaj sve dok ulazni karakter nije jedno od slova A, B ili C.
  - %[^]%^B-E+] - učitavaj dok ne učitaš], %, ^, slova od B do E ili +.
- Kad se navede specifikator opsega, ne navode se specifikatori formata!

Primjer

- Pretpostavimo sljedeće učitavanje podataka:

```
int x;
float y;
char ime[50];
scanf("%2d  %f  %*d  %[abck-m0-6]", &x, &y, ime);
```

- Ako unesemo 56789 123 m56bc723, rezultat učitavanja je:

```
x = 56
y = 789.0
ime = "m56bc"
```

- Preostali dio unosa ostaje u baferu i čeka sljedeću naredbu za unos podataka!

Zanemarivanje karaktera za prelazak u novi red

- Kada želimo da učitamo neki podatak (karakter, broj, string), a nakon toga karakter, na sljedeći način:

```
scanf("%s", str);
scanf("%c", &ch);
```

nakon unosa prvog podatka (ovdje stringa) i pritiska Enter, u promjenljivu ch će biti smješten karakter za novi red.

- Funkcija scanf prilikom učitavanja karaktera (ne i ostalih tipova!) ne uklanja bjeline! Ovo je moguće izbjeći na sljedeći način:

```
scanf("%[^\n]%*c", str);  // Učitavaj sve do Enter-a, pa zanemari Enter
scanf("%c", &ch);
```

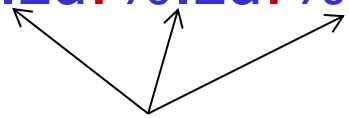
- Drugi način da se to izbjegne je da se ubaci razmak ispred %c:

```
scanf(" %c", &ch);  // ignoriši Enter posle %s, pa učitaj karakter
```

sprintf

- Pored pobrojanog, u stdio.h definisana je i znakovna konstanta EOF koja označava kraj fajla.
- Interesantna je i funkcija sprintf kojom se vrši štampanje u string. Slična je funkciji printf, osim što ima dodatni (prvi) argument string u koji se vrši "štampanje" umjesto na ekran.
- Na primjer, ako želimo da formiramo string koji predstavlja tekuće vrijeme u formatu "HH:MM:SS", a posjedujemo cijele brojeve koji predstavljaju sate, minute i sekunde, to možemo uraditi kao:

```
sprintf(s, "%.2d:%.2d:%.2d", sat, min, sek);
```



Tačka označava da se prazna mjesta dopunjavaju nulama

sscanf

- Za razliku od scanf, koja učitava podatke sa standardnog ulaznog uređaja, funkcija sscanf učitava podatke iz stringa koji je zadat kao prvi argument. Ostali argumenti su isti kao kod scanf.
- **Primjer:** Ako je u stringu s zadato vrijeme u sljedećem formatu s = "HH:MM:SS", onda funkcija

```
sscanf(s, "%2d%*c%2d%*c%2d", &sat, &min, &sek)
```

u promjenljive sat, min i sek učitava (upisuje) sate, minute i sekunde, respektivno. Pomoću %*c zanemarujemo dvotačke.

string.h - pregled i još ponešto

- Podsjetite se funkcija: strcmp, strcpy, strlen.
- Pored ovih, koriste se i funkcije:
  - strncpy(dest, source, n) kopira string source u string dest, ali najviše do n karaktera;
  - strncat(dest, source, n) nadovezuje string source na string dest, ali najviše do n karaktera;
  - strchr(s, c) vraća pokazivač na prvo pojavljivanje karaktera c u stringu s (ako ne nađe, vraća NULL);
  - memcmp(s1, s2, n) poredi sadržaj memorije (kao da su u pitanju stringovi), ali najviše do n bajtova.

string.h

- memcpy(s1, s2, n) kopira sadržaj n bajtova memorije sa pozicije s2 na poziciju s1 (s1 i s2 su pokazivači). Brža je od strcpy.
- memchr(s, c, n) traži prvo pojavljivanje "karaktera" c u memoriji koja počinje od pokazivača s, najviše do n bajtova (vraća NULL ako nema).
- memset(s, c, n) kopira karakter c na prvih n pozicija stringa s.
- strtok(s, delim) dijeli string s na podstringove razdvojene karakterima (delimiterima) u stringu delim (primjer ispod). Mijenja se string s!

```
char str[] = "Prva, druga, i treca: kuku-riku.";
char delim[] = " -,.:"; ← Karakteri delimiteri
char *podstr;
podstr = strtok(str, delim); ← Prvi poziv sa str
while(podstr != NULL) {
    printf("%s\n", podstr);
    podstr = strtok(NULL, delim);
}
```

Naredni pozivi sa NULL, počinje se od kraja prethodnog podstringa

Ispis:

```
Prva
druga
i
treca
kuku
riku
```

ctype.h

- Biblioteka ctype.h raspolaže sa velikim brojem funkcija za rad sa karakterima:
  - isdigit(c) vraća nenultu vrijednost ako karakter c predstavlja cifru i 0 u suprotnom;
  - isalnum(c) vraća nenultu vrijednost ako je c alfanumerički karakter (cifra ili slovo) i 0 u suprotnom;
  - isalpha(c) vraća nenultu vrijednost ako je c slovo i 0 u suprotnom.
  - isprint(c) je tačno ako se karakter može odštampati (postoje karakteri koji se ne mogu štampati, npr. '\a' koje daje alarm zvuk);
  - tolower(c) ako je karakter veliko slovo prebacuje se u malo;
  - toupper(c) ako je karakter malo slovo prebacuje se u veliko.

math.h

- Solidan broj matematičkih funkcija je realizovan u biblioteci math.h. Većinu nije potrebno detaljno obrazlagati: sqrt, sin, cos, asin, acos, tan, atan, sinh, cosh, ... Stepena funkcija je pow(x,y) koja daje x^y, dok funkcija atan2(x,y) vraća rezultat atan(x/y), ali u granicama od $-\pi$ do π.
- Postoje samo dvije funkcije za zaokruživanje: ceil, koja zaokružuje na veći cijeli, i floor, koja zaokružuje na manji cijeli. Kako se obavlja zaokruživanje ka 0?

limits.h

- Značaj ove biblioteke je prije svega u činjenici da raspolaže sa konstantama koje pomažu u izbjegavanju upotrebe mašinski zavisnih elemenata programa.
- Na primjer, konstante CHAR_MIN i CHAR_MAX predstavljaju minimalni i maksimalni cijeli broj koji odgovara tipu char, UCHAR_MAX je maksimalni unsigned char broj. Slično, postoje:
  - SHRT_MIN i SHRT_MAX za short int,
  - INT_MIN i INT_MAX za int,
  - LONG_MIN i LONG_MAX za long int.

limits.h i stdlib.h

- Slične granice postoje i za float (FLT_MIN i FLT_MAX) kao i za double (DBL_MIN i DBL_MAX).
- I drugim mašinski zavisnim elementima se može pristupiti koristeći ovu programsku biblioteku. Na primjer, FLT_DIG i DBL_DIG su simboličke konstante koje predstavljaju broj decimalnih mjesta kod float-a i double-a.
- Izuzetno važna programska biblioteka je stdlib.h. Funkcije itoa(N, s, B) i ltoa(N, s, B), koje konvertuju int N i long N u string s u brojnom sistemu sa osnovom B, se nalaze u ovoj biblioteci. Pomenimo još dvije funkcije: exit() i rand().

stdlib.h

- Funkcija rand() vraća cijeli broj, a = rand(), koji je na osnovu nekog algoritma odabran na intervalu od 0 do nekog cijelog broja, predstavljenog simboličkom konstantom RAND_MAX iz stdlib.h (sigurno ≥ 32767). Ova funkcija nema argumenata.
- rand() se naziva generatorom (pseudo-) slučajnih brojeva.
- Mnoge aplikacije, a posebno igrice, se ne mogu zamisliti bez ovog generatora. Na primjer, generisanje slučajnog broja na intervalu od 1 do 6, što odgovara bacanju kocke, se postiže sa a = rand()%6 + 1. Protumačite!
- Funkcija exit(a), gdje je a neki cijeli broj, prekida izvršavanje programa. Argument a je obavezan, on se prosljeđuje operativnom sistemu, ali sa našeg stanovišta njegova vrijednost nije bitna, pa ćemo stavljati exit(1).

Karakteristike promjenljivih

- Za nekoga ko je učio 6-7 nedjelja programiranje ponovno upoznavanje sa karakteristikama promjenljivih može djelovati nepotrebno.
- Ipak, itekako je potrebno!
- Naime, osnovni pojmovi o promjenljivim koje smo do sada koristili nijesu i jedine važne činjenice o njima.
- U okviru ovog časa ćemo se osvrnuti na neke od njih.
- Sve promjenljive imaju dvije karakteristike:
  - opseg (dio programskog koda koji može da pristupi promjenljivoj) i
  - trajanje (vrijeme koje promjenljive provedu u memoriji).

Lokalne i globalne promjenljive

- Do sada smo promjenljive deklarisali na početku funkcija.
- Ovo su bile lokalne promjenljive kojima je opseg funkcija u kojoj su definisane, a trajanje ograničeno vremenom izvršavanja funkcije u kojoj su definisane.
- Promjenljive se mogu deklarisati na početku svakog bloka naredbi. Ovo su, takođe, lokalne promjenljive vidljive do kraja svog bloka.
- Nakon napuštanja svog bloka naredbi, promjenljive se dealociraju (brišu iz memorije).
- Pored ovoga postoje i globalne promjenljive.
- Globalne promjenljive se definišu van bilo koje funkcije (obično prije svih), vidljive su iz svih funkcija i traju do kraja programa.
- Ako se globalne promjenljive ne inicijalizuju, vrši se podrazumjevana inicijalizacija na 0.

Primjer lokalnih i globalnih prom.

```
int x=0;        // globalna promjenljiva
void fun(int z)
{int y;}        // vidljivi lokalna y, argument z (isto lokalna) i globalna x

int main() {
    int y;      // vidljiva lokalna y i globalna x
    {
       int z;   // vidljiva lokalna iz bloka z, lokalna iz funkcije y, i globalna x
    }
    ...         // vidljiva lokalna y i globalna x,  z je dealocirana
}
```

Problem može predstavljati situacija kada unutar nekog bloka imamo promjenljivu koja se zove isto kao promjenljiva iz spoljašnjeg bloka.

Zasjenjivanje

- Deklaracija unutar bloka promjenljive sa istim imenom kao što je neka promjenjiva, koja bi inače bila vidljiva unutar tog bloka, naziva se **zasjenjivanje**.
- Zapamtite da nije dozvoljeno deklarisati promjenljivu u istom bloku dva puta, ali je dozvoljena deklaracija u više blokova.

```
int x=0;          // globalna promjenljiva
int main() {
    int x=1;      // vidljiva lokalna x=1
    {
        int x=2; // vidljiva lokalna iz bloka x=2
    }
    ...           // vidljiva lokalna x=1. x=2 je dealocirana
}
```

Iako nose isto ime, imamo tri različite promjenljive **x**.

Statičke promjenljive

- Promjenljive se mogu podijeliti i na:
  - dinamičke i
  - statičke.
- Dinamičke se dealociraju pri napuštanju bloka u kome su definisane. Sve do sada deklarisane lokalne promjenljive su dinamičke.
- Statičke promjenljive traju tokom čitavog izvršavanja programa (možda im se u nekom trenutku ne može pristupiti zbog zasjenjivanja ili drugih razloga, ali postoje).
- Globalne promjenljive su statičke!

Statičke promjenljive

- Lokalne promjenljive se mogu učiniti statičkim ako se doda ključna riječ **static** prilikom njihovog deklarisanja. Ako se ne inicijalizuju, dodjeljuje im se podrazumijevana vrijednost 0.
- Često se koriste kod funkcija.

```
void funk() {
    static int i = 100;
    i++;
    printf("%d ", i);
}
int main() {
    int i;              // ovo nije ista promjenljiva kao statička iz funk()
    for(i = 0; i < 3; i++)   funk();
}
```

Iznenađujuće, ali na ekranu će biti ispisano: 101 102 103

Statičke promjenljive

- Kod prvog poziva funkcije sve je jasno. Vrijednost i=101, ali se ne dealocira nakon napuštanja funkcije.
- U narednom pozivu, preskače se deklaracija statičke promjenljive, jer se jedna promjenljiva ne može više puta deklarisati, pa se i uvećava na 102. Dalje je sve jasno.
- Statičke promenljive se moraju inicijalizovati konstantom!
- Korišćenje globalnih i statičkih promjenljivih može voditi ka izuzetno elegantnim rješenjima, ali i do veoma teško razumljivog koda koji je nepogodan za održavanje.
- Prije upotrebe statičkih i globalnih promjenljivih barem dva puta razmislite.

register i volatile promjenljive

- Sve promjenljive koje su do sada uvedene su se mogle deklarisati sa dodatnim modifikatorom - auto. Pošto je ovaj modifikator podrazumjevan, mi smo ga izostavljali (automatski smještaj promjenljivih podrazumjeva alokaciju u memoriji).
- Postoji mogućnost da promjenljive budu deklarisane sa modifikatorom register.
- Ovaj modifikator forsira smještaj promjenljive u registre procesora.
- Pristup registrima procesora je veoma brz, ali je slobodnih registara malo.

register i volatile promjenljive

- Ako smještaj u registre nije moguć promjenljiva će biti smještena u memoriju bez obavještenja.
- Od registarskih promjenljivih se ne može uzeti adresa!
- Ako se koriste register promjenljive, po nekom nepisanom pravilu to su brojači ili promjenljive koje se u ciklusima intenzivno koriste.
- Vrlo zagonetan tip promjenljivih su **volatile** promjenljive.
- Deklarisati promjenljivu kao **volatile** znači ukazati kompajleru da ova promjenljiva može biti bilo kad promijenjena "spolja" nekim drugim programom, radom operativnog sistema ili čak instrukcijama koje imaju veze sa periferijama računara.
- Kompajler zatim izbjegava optimizaciju djelova koda sa volatile modifikatorom.

Eksterne promjenljive

- volatile i register promjenljive nećemo koristiti u našim programima. One se uglavnom koriste u embedded programiranju.
- Programski kôd je često izdijeljen u više fajlova.
- Ti fajlovi se prevode do mašinskog programa u relativnim adresama (OBJ verzije kod našeg kompajlera), pa se zatim samo zajedno povežu u izvršni EXE fajl (sa apsolutnim adresama).
- Postoje situacije kada se globalna promjenljiva koja je definisana u jednom fajlu mora koristiti i u drugim fajlovima.
- Njeno postojanje se tada mora najaviti u drugim fajlovima.

Eksterne promjenljive

- Ako je u fajlu **F1.c** deklarisana globalna int x, a želimo je koristiti u fajlu **F2.c**, moramo je u **F2.c** najaviti kao extern int x. Ovo nije deklaracija, već najava da koristimo globalnu promjenljivu deklarisanu u nekom drugom fajlu.
- U fajlu F2.c možemo mijenjati globalnu int x iz F1.c.
- Ukoliko je globalna promjenljiva deklarisana kao static, ne može joj se pristupiti van fajla gde je deklarisana! Ovo je mehanizam čuvanja privatnosti globalnih promjenljivih.
- Isto važi za static funkcije.
- U velikim programskim paketima ponekad postoji potreba za globalnim promjenljivim koje će koristiti svi programski moduli, ali u našim programima to je rijetko potrebno.

Dinamička alokacija i dealokacija

- Do sada smo alocirali nizove na sljedeći način:
  int a[50];
- Unutar zagrada smo upisivali najveći mogući broj elemenata niza koji se u datom programu može pojaviti i to je tzv. **statička alokacija memorije**.
- Neracionalno je zauzeti 1000 memorijskih pozicija za najgori slučaj, kada će, na primjer, biti rađeno sa nizovima koji imaju nekoliko desetina članova, samo zbog toga se što kod nekih izuzetno zahtjevnih radnji može pojaviti 1000 članova.
- Stariji programski jezici koji su vršili alokaciju memorije statički (samo jednom u sekciji za deklaraciju) morali su da rade na ovaj način.

Dinamička alokacija i dealokacija

- Savremeni programski jezici mogu da izvrše alokaciju memorije dinamički tokom rada programa.
- Da bi kompajler programskog jezika C mogao da vrši dinamičku alokaciju mora biti uključeno zaglavlje stdlib.h.
- Ako želimo da alociramo niz cijelih brojeva a dinamički, na početku programa ćemo deklarisati samo pokazivač a:

  int *a;
- Kada saznamo sa koliko podataka korisnik želi da radi, možemo izvršiti dinamičku alokaciju pomoću jedne od funkcija za to, a to je najčešće funkcija **malloc**.

Upotreba funkcije malloc

```
a = (int *)malloc(N*sizeof(int));
```

- Funkcija malloc zauzima memoriju za određen broj bajtova (argument ove funkcije je veličina memorije u bajtovima). Ako se želi zauzeti memorija za N cijelih brojeva i ako se žele izbjeći mašinski zavisni elementi, kao argument se koristi N*sizeof(int).
- Funkcija malloc vraća pokazivač na void (void *), koji pokazuje na početak bloka zauzete memorije.
- Da bi taj pokazivač pokazivao na odgovarajući tip moramo ga primjenom cast operatora pretvoriti u pokazivač na željeni tip (to je ovdje urađeno sa (int *)).

Upotreba funkcije malloc

- Ako se zauzima memorija za neki drugi tip podatka, mijenja se samo cast operator ispred funkcije malloc i "argument" operatora sizeof u argumentu funkcije.
- Postoji mogućnost da zbog skučenih resursa računara nije moguće izvršiti dinamičku alokaciju.
- Tada malloc vraća NULL pokazivač.
- Svaka dinamička alokacija mora obavezno biti praćena provjerom da li je operacija uspjela i ako nije treba preduzeti korektivne akcije ili prosto izaći iz programa.

Upotreba funkcije malloc

- Primjer kako treba koristiti funkciju malloc:

```
a = (int *)malloc(N*sizeof(int));
if(a==NULL)   exit(1);
```

- Ako alokacija nije uspjela, izlazimo iz programa, što je jedna od mogućnosti.
- Ako je alokacija niza uspjela mi koristimo elemente niza a[0], a[1], ..., a[N-1] na isti način kao da su statički alocirani.
- Nakon posljednje upotrebe elemenata niza, a prije napuštanja programa, potrebno je izbrisati (dealocirati) memoriju koja je dinamički zauzeta.

Dealokacija i druge funkcije za alokaciju

- Dealokacija se obavlja funkcijom **free(a)**, a argument ove funkcije je pokazivač na memorijski blok koji je zauzet funkcijom malloc.
- Funkcija **free** nije praćena provjerom uspješnosti dealokacije.
- Pored funkcije malloc u zaglavlju stdlib.h su definisane još neke za dinamičku alokaciju.
- Prva od njih je **a = calloc(N, sizeof(int));** koja ima isti smisao kao malloc(N*sizeof(int)) (zauzima memoriju za N podataka koji imaju veličinu sizeof(int) bajtova). Jedina je razlika što calloc inicijalizuje zauzetu memoriju na 0.

realloc

- Funkcija **realloc** vrši promjenu veličine bloka memorije koji je pridružen pokazivaču (obično se blok povećava, mada ima situacija kada treba raditi suprotno).
- Poziv funkcije **realloc** ima sljedeći oblik:

  realloc(a, n)

  gdje je a pokazivač na već alociranu memoriju, dok je **n** veličina memorije (može i n*sizeof(int)) na koju treba nakon naredbe da pokaže a.
- I funkcije **calloc** i **realloc** zahtjevaju provjeru poređenjem sa **NULL**.

Ostatak gradiva

- Završili smo sa svim osnovnim elementima programskog jezika C.
- Ostatak našeg kursa vezan je za složene tipove podataka koji postoje u programskom jeziku C:
  fajl, nabrajanje, struktura (sa poljem bitova) i unija, kao i sa tipovima podataka koji se mogu kreirati pomoću već definisanih tipova u programskom jeziku C: liste, grafovi, stabla (drveta).