

Programiranje 2

Predavanje 05 - dinamička alokacija memorije, stringovi

Matej Mihelčič

Prirodoslovno-matematički fakultet
Matematički odsjek

31. ožujka 2025.



Tri vrste objekata prema načinju rezervacije memorije:

- **automatski** - rezervacija pri svakom ulasku u blok
- **statički** - rezervacija jednom, na početku izvršavanja programa
- **dinamički** - kreiramo po potrebi tijekom izvršavanja programa koristeći dinamičko rezerviranje (alokaciju) memorije.

Dinamičke objekte možemo uništiti oslobađanjem alocirane memorije.

Dinamička alokacija memorije služi za kreiranje polja kod kojih dimenzija nije unaprijed poznata, dinamičkih struktura podataka (npr. vezane liste, stabla). Dinamički objekti se spremaju u bloku memorije koji se zove **hrpa** (eng. runtime heap).

Funkcije za alokaciju i dealokaciju memorije deklarirane su u datoteci zaglavlja `<stdlib.h>` (standardna biblioteka).

- alokacija: funkcije `malloc`, `calloc`, `realloc`.
- dealokacija: funkcija `free`.

Memoriju možemo dinamički alocirati funkcijom malloc.

```
1 void *malloc(size_t n);
```

size_t je cjelobrojni tip bez predznaka (za spremanje veličina objekata) definiran u <stddef.h>, a n je jednak **ukupnom broju bajtova** koji treba alocirati.

Funkcija malloc rezervira blok memorije od n bajtova. Vraća pokazivač na rezervirani blok memorije, ili NULL, ako se zahtjev ne može ispuniti. Vraćeni pokazivač je generički, tipa void*. Prije upotrebe ga treba eksplicitno konvertirati u potrebni tip pokazivača (cast operatorom).

Druga mogućnost za dinamičku alokaciju memorije je funkcija `calloc`.

```
1 void *calloc(size_t nobj, size_t size);
```

Funkcija `calloc` rezervira blok memorije za spremanje `nobj` objekata, od kojih svaki pojedini objek ima veličinu `size`, tj. ukupan broj rezerviranih bajtova je `nobj * size`. Dodatno, inicijalizira cijeli rezervirani prostor na nule, preciznije na nul-znakove `'\0'`. Kao i `malloc`, vraća pokazivač na rezervirani blok ili `NULL`.

Alokacija memorije za 150 elemenata tipa double:

```
1 double *p;  
2 ...  
3 p = (double *) malloc(150 * sizeof(double));  
4 if (p == NULL) {  
5     printf("Greska: alokacija nije uspjela!\n");  
6     exit(EXIT_FAILURE); /* exit(1); */  
7 }
```

Možemo koristiti i calloc koji inicijalizira vrijednosti na nulu.

```
1 p = (double *) calloc(150, sizeof(double));
```

Kod dinamičke alokacije memorije uvijek treba provjeriti je li alokacija uspjela: `if (p == NULL) ...`. U slučaju neuspjele alokacije treba **prekinuti** izvršavanje programa. Program prekidamo korištenjem funkcije `exit` deklarirane u biblioteci `<stdlib.h>`.

```
1 void exit(int status);
```

Poziv `exit(status);` zaustavlja izvršavanje programa i vrijednost `status` predaje operacijskom sustavu, tj. radi isto što i `return status;` u funkciji `int main`. Funkciju `exit` možemo koristiti u bilo kojoj funkciji. `status ≠ 0` signalizira grešku.

Alociranu memoriju, nakon upotrebe, možemo osloboditi funkcijom `free`.

```
1 void free(void *p);
```

Funkcija `free` uzima pokazivač p na početak alociranog bloka memorije i oslobađa taj blok memorije. Ako je $p == \text{NULL}$, onda ne radi ništa. Funkcija `free` **ne mijenja** pokazivač p . Nakon poziva `free(p)`; pokazivač p i dalje pokazuje na isti (oslobođeni) dio memorije i taj dio (tj. $*p$) se ne smije koristiti. Dobra praksa je iza poziva `free(p)`; staviti $p = \text{NULL}$;

Dinamičko kreiranje polja

Program dinamički kreira polje a brojeva tipa `int`, a broj n elemenata polja se učitava. Ispisuje zbroj svih elemenata u polju.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(void){
5
6 int *a; /* Pokazivac na dinamičko polje. */
7 int i, n, zbroj;
8 printf("Upisi broj elemenata polja a:");
9 scanf("%d", &n);
10
11 if ((a = (int*) calloc(n, sizeof(int)))== NULL) {
12     printf("Alokacija nije uspjela.\n");
13     exit(EXIT_FAILURE);} /* exit(1); */
```

Dinamičko kreiranje polja

```
14 for (i = 0; i < n; ++i) {
15     printf("Upisi element polja a[%d]: ", i);
16     scanf("%d", &a[i]); }
17
18 zbroj = 0;
19
20 for (i = 0; i < n; ++i)
21     zbroj = zbroj + a[i];
22
23 printf("Zbroj svih elemenata = %d\n", zbroj);
24 free(a); /* Ne treba a = NULL; kraj programa. */
25 return 0; }
```

Treća mogućnost za dinamičku alokaciju memorije je funkcija `realloc`. Služi za promjenu veličine već alociranog bloka.

```
1 void *realloc(void *p, size_t size);
```

Funkcija `realloc` mijenja veličinu objekta na kojeg pokazuje p na zadanu veličinu `size` (tj. realocira memoriju). Sadržaj objekta ($*p$) ostaje isti do minimuma stare i nove veličine (kopira se po potrebi). Ako je nova veličina objekta veća od stare, dodatni prostor se ne inicijalizira. Vraća pokazivač na novorezervirani prostor, ili `NULL`, ako zahtjev nije ispunjen (tada $*p$ ostaje nepromijenjen).

`realloc`-om uglavnom produljujemo dinamičke objekte (polja) nepoznate duljine.

Moguća su dva scenarija:

- Duljinu niza n učitamo i rezerviramo memoriju za cijeli niz.
- Broj članova ne znamo unaprijed, čitamo nepoznati broj članova do oznake za kraj niza. Pri dodavanju svakog novog člana niza produljujemo postojeći niz za po jedan član, počevši od praznog niza.

Isti princip možemo iskoristiti kod kreiranja rječnika (niza stringova).

```
1 int *p = NULL, i;  
2  
3 p = (int*) realloc(p, 10 * sizeof(int));  
4  
5 for(i=0; i<10; i++)  
6     p[i] = 2*i;
```

Realokacija memorije - realloc

```
1  int *p = NULL, n, i, br;
2  scanf("%d",&n);
3
4  if(n<0){ exit(0); }
5  p = (int*) realloc(p,sizeof(int));
6  br = 1; p[0] = n;
7
8  while(n>=0){
9      scanf("%d",&n);
10     if(n<0){
11         for(i=0;i<br;i++)
12             printf("%d□",p[i]);
13         printf("\n");
14         free(p); exit(0); }
15     p = (int*) realloc(p,++br*sizeof(int));
16     p[br-1] = n; }
```

Kod rada s vektorima (poljima), matricama (višedimenzionalnim poljima), čak i kada u programu koristimo više objekta, uglavnom dimenzije ne variraju.

Za razliku od toga, kod obrade teksta imamo riječi čije veličine jako variraju. Da bi mogli učinkovito raditi s nizovima riječi, trebali bi pamtit i sve njihove duljine što nije praktično.

Rješenje: uvesti poseban znak koji će označavati kraj radnog dijela niza.

Realizacija takvog niza u C-u se zove **string**.

```
1 "Jedan_primjer_stringa."
```

Ideja posebne oznake za kraj se koristi i kod **vezane liste**.

Polje znakova ili niz znakova je bilo koje polje znakova.

```
1 char poruka[128]; // Polje od 128 znakova.
```

String je polje (niz) znakova koje sadrži bar jedan nul-znak '\0'. Prvi nul-znak (onaj s najmanjim indeksom) ima ulogu oznake za kraj niza (radni sadržaj se nalazi ispred tog znaka).

Svaki string je polje znakova. Razlika je u interpretaciji sadržaja. Korištenjem formata %s, sadržaj se interpretira kao string, međutim tada mora sadržavati nul-znak.

Duljina stringa je **broj znakova** ispred nul-znaka, tj. duljina **radnog sadržaja** stringa. Osnovna funkcija za rad sa stringovima je funkcija `strlen` koja vraća duljinu zadanog stringa. Deklarirana je u datoteci zaglavlja `<string.h>`. Prototip (zaglavlje) funkcije `strlen` je:

```
1 size_t strlen(const char *s)
```

Vraća duljinu stringa `s` bez oznake za kraj niza, znaka `'\0'`. String je zadan **pokazivačem** na (konstantni) **prvi znak**, tj. funkcija **ne smije** promijeniti sadržaj stringa.

Inicijalizacija polja - niz znakova i string

Definicija (nije string, ne sadrži nul-znak, ponašanje funkcija za stringove nije definirano):

```
1 char niz[4] = {'a', 'b', 'b', 'a'};
```

kreira niz od 4 elementa, `sizeof(niz) = 4`. `niz[5]` bi bio string.

'a'	'b'	'a'	'b'
-----	-----	-----	-----

Definicije (string, sadrži nul-znak):

```
1 char s[5] = {'a', 'b', 'b', 'a', '\0'};  
2 char s[5] = "abba";
```

kreiraju polje od 5 elemenata, `sizeof(s) = 5`. `strlen(s) = 4`.

'a'	'b'	'a'	'b'	'\0'
-----	-----	-----	-----	------

Zanima nas korištenje i razlika između:

```
1 char a[] = "Poruka"  
2 char *p = "Poruka";
```

- Pridruživanje konstantnog stringa polju je moguće **samo prilikom inicijalizacije**. Smijemo mijenjati **vrijednosti** unutar polja, ali ne i vrijednost pokazivača *a*.
- Pridruživanje konstantnog stringa pokazivaču smijemo vršiti pri inicijalizaciji i korištenjem operatora pridruživanja. **Ne smijemo** mijenjati sadržaj stringa (string je konstantan, rezultat izmjene je nedefiniran). Međutim, možemo mijenjati vrijednost pokazivača (smije pokazivati na neki drugi string).

Pridruživanje stringa

Na lijevoj strani naredbe pridruživanja smije biti bilo koji objekt koji nije konstantan (*lvalue* izraz). To znači da je dozvoljeno promijeniti vrijednost tog objekta. Isto vrijedi i za pokazivač na `char`. Ako nije konstantan, smijemo mu pridružiti i konstantni string.

```
1 char *p;  
2 ...  
3 p = "Poruka";
```

```
1 char polje[10], s[5];  
2 ...  
3 polje = "Poruka"; /* Pogresno */  
4 s = "tri"; /*Pogresno */  
5 /*mozemo koristiti strcpy*/  
6 strcpy(polje, "Poruka");  
7 strcpy(s, "tri");
```

Stringovi (kao i polja) su zadani pokazivačem na prvi element (znak).

```
1 char s1[] = "Dobar_dan", s2[] = "Dobar_dan";
2 ...
3 if (s1 == s2) printf("jednaki");
4 else printf("razliciti");
```

Rezultat gornje usporedbe je **različiti** jer se uspoređuju **adrese** stringova (pokazivači na prve elemente) koji su različiti.

Usporedbu stringova **znak po znak** radi funkcija `strcmp`.

Čitanje stringova se vrši funkcijom `scanf` koristeći format:

- `%s` - string omeđen bjelinama
- `%[...]` i `%[^...]` - string koji sadrži ili ne sadrži (drugi slučaj) navedene znakove.

Funkcija `gets` (ili `gets_s`) - čita cijeli red uz `'\n'` → `'\0'`.

U funkciji `scanf` uvijek treba zadati maksimalnu širinu (duljinu) polja. Također, treba koristiti `fgets` (`gets_s`) a ne `gets` (izbačena iz jezika od verzije C11).

Pisanje stringova se vrši funkcijom `printf` korištenjem formata `%s` - string bez završnog `'\0'`.

Funkcija `puts` - ispiše cijeli red uz `'\0'` → `'\n'`.

Čitanje i pisanje stringova

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5
6     char string[80];
7     scanf("%79s", string);
8     /* scanf("%79[^\n]", string); */
9
10    puts(string);
11    printf("_Duljina_str._=%u\n", strlen(string));
12    return 0; }
```

Ulaz: Dobar dan\n

Izlaz: za `scanf("%79s", string):`

Dobar

Duljina str. = 5

Izlaz: za `scanf("%79[^\n]", string):`

Dobar dan

Duljina str. = 9

Kod čitanja s `gets` ili `gets_s` učitava se i **kraj linije**.

C sadrži mnogo funkcija za:

- obradu stringova - u datoteci zaglavlja `<string.h>`
- obradu znakova - u datoteci zaglavlja `<ctype.h>`

Kod dinamičke alokacije memorije možemo koristiti aritmetiku pokazivača ili koristiti indekse. To se često koristi kod obrade stringova.

Želimo napisati funkciju `invertiraj` koja invertira ulazni string. String je zadan pokazivačem na prvi element.

Za razliku od prosljeđivanja polja funkciji, gdje kao dodatni parametar trebamo proslijediti i duljinu polja, kod stringa ne prosljeđujemo duljinu polja već sami tražimo kraj ili koristimo funkciju `strlen`.

Pokazat ćemo tri varijante funkcije: preko polja koristeći indekse (dva načina) i preko pokazivača (koristeći aritmetiku pokazivača).

Invertiranje stringa

```
1 void invertiraj(char s[]){
2   int p, k; /* p = pocetak, k = kraj. */
3   char temp;
4
5   for (p = 0, k = strlen(s)-1; p < k; ++p, --k) {
6     temp = s[p];
7     s[p] = s[k];
8     s[k] = temp; }
9
10  return;
11 }
```

Invertiranje stringa

```
1 void invertiraj(char s[]){
2 int i, n = strlen(s);
3 char temp;
4 /* Ne koristiti strlen u petlji. */
5 for (i = 0; i < n/2; ++i) {
6     temp = s[i];
7     s[i] = s[n - 1 - i];
8     s[n - 1 - i] = temp;
9 }
10 return;
11 }
```

Invertiranje stringa

```
1 void invertiraj(char *s){
2 char temp, *p, *k; /* p = pocetak, k = kraj. */
3 p = s; k = p + (strlen(s) - 1);
4
5 while (p < k) {
6     temp = *p;
7     *p++ = *k;
8     *k-- = temp;
9 }
10 return;
11 }
```

U kodu iznad prvo se na pokazivačima izvršavaju ++/--, zatim operator *. Postfix oblik vraća pokazivač na staru lokaciju (prije inkrementiranja pokazivača).

Invertiranje stringa

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void invertiraj(char *s); /* ili (char s[]) */
5
6 int main(void) {
7     char str[80]; /* Nije inicijaliziran! */
8     /* Citanje i provjera za prazan string. */
9     if (scanf("%79[^\n]", str) < 1) str[0] = '\0';
10    printf("String (duljine %u):\n", strlen(str));
11    puts(str);
12    invertiraj(string);
13    printf("Invertirani string:\n");
14    puts(string);
15    return 0; }
```

Datoteka zaglavlja <string.h>

Za obradu stringova standardno se koriste funkcije deklarirane u datoteci zaglavlja <string.h>.

Funkcija `strlen`:

```
1 size_t strlen(const char *s)
```

vraća duljinu stringa `s` bez završnog `'\0'` znaka. String je zadan pokazivačem `s` na konstantni prvi znak (`*s`), tj. funkcija ne smije promijeniti sadržaj stringa preko pokazivača `s`.

Funkcija `strcpy`:

```
1 char *strcpy(char *s, const char *t)
```

kopira string `t` u string `s` (uključujući i završni `'\0'`) i vraća string pokazivač na prvi znak iz `s`.

Datoteka zaglavlja <string.h>

Funkcija strcat:

```
1 char *strcat(char *s, const char *t)
```

nadovezuje (konkatenira) string t na kraj stringa s , te vraća s . Prvi znak iz t kopira se na mjesto završnog nul-znaka `'\0'` u stringu s sve do kraja stringa t (uključujući `'\0'`).

Funkcija strcmp:

```
1 int strcmp(const char *s, const char *t)
```

leksikografski uspoređuje stringove s i t . Vraća broj:

- < 0 , ako je $s < t$
- $= 0$, ako je $s = t$
- > 0 , ako je $s > t$

Ponekad se `strcmp` implementira tako da je izlazna vrijednost razlika znakova na prvoj poziciji na kojoj se stringovi razlikuju (ako takvo postoji).

Npr. `strcmp("BAB", "BCB") = -2` jer je `'A' - 'C' = -2`.

Funkcije `strchr`, `strstr`:

```
1 char *strchr(const char *s, const int c)
2 char *strstr(const char *s, const char *t)
```

vraćaju pokazivač na znak koji je početak prvog pojavljivanja znaka `c` (za `strchr`), odnosno, stringa `t` (za `strstr`), u stringu `s` ako takvo mjesto postoji. U protivnom, ako ne postoji takvo mjesto vraćaju `NULL`.

Primjer funkcija iz <string.h>

Program zasebno čita ime i prezime, svako u svom redu, uz preskakanje bjelina na početku reda. Zatim ih spaja u jedan string, s prazninom između njih, i ispisuje taj string.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5     char ime[128], prezime[128];
6     char ime_i_prezime[256];
7
8     printf("Unesite ime:");
9     scanf("%127[^\n]", ime);
10
11    printf("Unesite prezime:");
12    scanf("%127[^\n]", prezime);
```

Primjer funkcija iz <string.h>

```
13  /* Spoji ime i prezime u jedan string */
14  strcpy(ime_i_prezime, ime);
15  strcat(ime_i_prezime, "_");
16  strcat(ime_i_prezime, prezime);
17  printf("Ime_i_prezime:_%s\n", ime_i_prezime);
18  return 0;
19  }
```

Ulaz: Marko Petar\n

Perić\n

Izlaz: Marko Petar Perić\n

Funkcije za obradu znakova deklarirane su u `<ctype.h>`. Sve funkcije imaju jedan argument tipa `int`, koji smije biti:

- znak EOF, standardno `EOF = -1` (zato se koristi `int`).
- znak prikaziv kao `unsigned char` (standardni znak).

Izlazna vrijednost je tipa `int`.

Funkcije iz `<ctype.h>` možemo podijeliti u dvije grupe:

- funkcije za provjeru znakova, vraćaju `int` različit od nule (istina), ako ulazni znak pripada određenoj grupi znakova. U protivnom vraćaju nulu (laž).
- funkcije za pretvaranje znakova, vraćaju pretvoreni ulazni znak.

Funkcije za provjeru znakova:

```
1 int isalpha(int c); /* Slovo, malo ili veliko */
2 int isdigit(int c); /* Numer. = dec. znamenka */
3 int isalnum(int c); /* Alfnumericki */
4 int isxdigit(int c); /* Heksadecimalna znam. */
5 int islower(int c); /* Malo slovo */
6 int isupper(int c); /* Veliko slovo */
7 int iscntrl(int c); /* Kontrolni znak */
8 int isgraph(int c); /* Ispisiv, bez blanka */
9 int isprint(int c); /* Ispisiv, uklj. blank */
10 int ispunct(int c); /* Ispisiv, bez blanka,
11 slova i dec. znamenki */
12 int isspace(int c); /* Bl, \n, \t, \v, \f, \r */
```

Funkcije iz <ctype.h>

U 7-bitnom ASCII kodu (0 do 0x7F, ili 0 do 127): ispisivi znakovi su: 0x20 (' ', tj. blank) do 0x7E ('~'), kontrolni znakovi su: 0 (NUL) do 0x1F (US) i 0x7F (DEL).

```
1 isdigit('0') = 1; isdigit('C') = 0;  
2 isalpha('0') = 0; isalpha('C') = 1;  
3 isxdigit('0') = 1; isxdigit('C') = 1;
```

Funkcije za pretvaranje mijenjaju samo slova:

```
1 int tolower(int c); /* Velika u mala */  
2 int toupper(int c); /* Mala u velika */
```

Primjer - implementacije nekih funkcija

Funkcija isdigit:

```
1 int isdigit(int c) {  
2     return ('0' <= c && c <= '9');  
3 }
```

Funkcija isalpha:

```
1 int isalpha(int c) {  
2     return ('a' <= c && c <= 'z' || 'A' <= c  
3         && c <= 'Z');  
4 }
```

Funkcija toupper:

```
1 int toupper(int c) {  
2     return ('a' <= c && c <= 'z') ?  
3         ('A' + c - 'a') : c; }
```

Primjena - strtoupper za stringove

Treba napisati funkciju strtoupper koja sva mala slova u zadanom stringu pretvara u velika.

```
1 void strtoupper(char *s){
2   int i;
3
4   for (i = 0; s[i] != '\0'; ++i)
5       if (islower(s[i]))
6           s[i] = toupper(s[i]);
7   return;
8 }
```

Primjena - strtoupr za stringove

Glavni program:

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4
5 void strtoupr(char *s);
6
7 int main(void) {
8     char kolegij[] = "Programiranje_2";
9     printf("_Pocetni_string:\n");
10    puts(kolegij);
11    strtoupr(kolegij);
12    printf("_Obradjeni_string:\n");
13    puts(kolegij);
14    return 0;
15 }
```

Funkcija samoglasnik provjerava je li zadani znak samoglasnik.

```
1 #include <ctype.h>
2 int samoglasnik(int c){
3     /*Pretvori c u malo slovo za ubrzanje provjere.*/
4     c = tolower(c);
5     return (c == 'a' || c == 'e' || c == 'i' ||
6     c == 'o' || c == 'u');
7 }
```

Funkcija suglasnik provjerava je li zadani znak suglasnik. Znak je suglasnik ako i samo ako je slovo i nije samoglasnik.

```
1 int suglasnik(int c){
2  /* Koristi funkciju samoglasnik
3  za ubrzanje provjere. */
4  return (isalpha(c) && !samoglasnik(c));
5 }
```

Primjer - samoglasnici u stringu

Funkcija vraća broj samoglasnika u stringu, kroz varijabilni argument vraća prvi samoglasnik u stringu (ukoliko postoji, inače ga ne mijenja).

```
1 int samogls(char *s, char *p_prvi){
2 int broj = 0, i;
3
4 for (i = 0; s[i] != '\0'; ++i)
5 if (samoglasnik(s[i]))
6     if (++broj == 1) *p_prvi = s[i];
7 return broj;
8 }
```

Primjer - strlen

```
1 int strlen(const char *s){
2 int n;
3
4 for (n = 0; s[n] != '\0'; ++n);
5 /* for je prazan! */
6 return n;
7 }
```

```
1 int strlen(const char *s){
2 int n = 0;
3 //mozemo implementirati i
4 //koristenjem dva pokazivaca
5 while (*s++ != '\0') /* (*s++) */
6     ++n;
7 return n;
8 }
```

Primjer - strcpy

```
1 void strcpy(char *s, char *t){
2 int i = 0;
3 while ((s[i] = t[i]) != '\0') ++i;
4 return;
5 }
```

```
1 void strcpy(char *s, char *t){
2 while ((*s = *t) != '\0') {
3     ++s; ++t; }
4 return;
5 }
```

```
1 void strcpy(char *s, char *t){
2 while ((*s++ = *t++) != '\0');
3 return;
4 }
```

Primjer - strcat

```
1 char *strcat(char *dest, const char *src){
2 char *p = dest;
3 while (*p++); /* Pomak do IZA '\0' u dest. */
4 --p; /* Natrag na zadnji '\0'. */
5 while (*p++ = *src++); /* Kopiraj src u dest. */
6 return dest;
7 }
```

```
1 char *strcat(char s[], const char t[]){
2 int i = 0, j = 0;
3 while(s[i] != '\0') ++i;
4 while((s[i++] = t[j++]) != '\0');
5 return s;
6 }
```