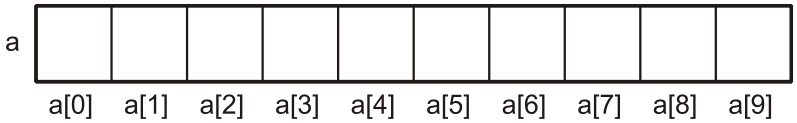
Pokazivači i nizovi

- U C-u postoji jaka veza između pokazivača i nizova
- Deklaracija

```
int a[10];
```

definiše niz a veličine 10, odnosno niz od 10 uzastopnih celih brojeva a[0], a[1], ..., a[9].



Shematski prikaz niza a u memoriji računara.

- Notacija `a[i]` ukazuje na i-ti element niza.

Pokazivači i nizovi

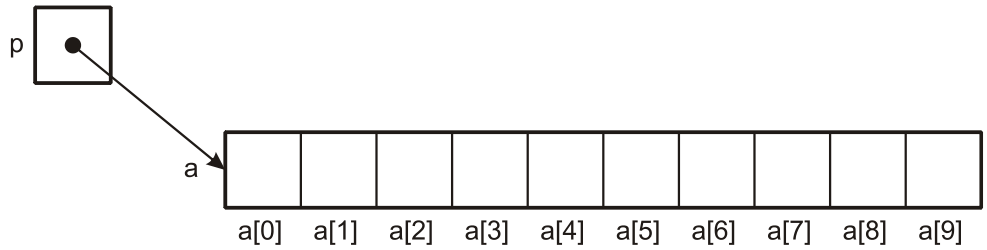
- Ukoliko je p pokazivač na ceo broj, deklarisan kao

```
int *p;
```

onda linija

```
p = &a[0];
```

postavlja pokazivač **p** da pokazuje na nulti element niza **a** ili, drugačije rečeno, pokazivač **p** sadrži adresu elementa **a[0]**.



Pokazivač p pokazuje na nulti element niza a

Pokazivači i nizovi

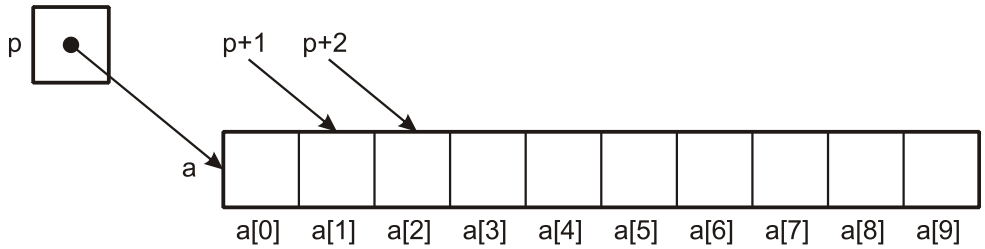
- Sada linija

  `x = *p;`

  dodeljuje promenljivoj x vrednost elementa `a[0]`.
- Ukoliko pokazivač p pokazuje na određeni element niza, po definiciji
  - `p+1` pokazuje na sledeći element,
  - `p+i` pokazuje na i-ti element iza p, a
  - `p-i` pokazuje na i-ti element ispred p.
- Tako, ukoliko p pokazuje na `a[0]`, onda

  `*(p+1)` predstavlja sadržaj elementa `a[1]`, a

  `*(p+i)` sadržaj elementa `a[i]`.



- Prethodne napomene važe bez obzira na tip ili veličinu promenljivih u nizu a

Pokazivači i nizovi

- Indeksiranje nizova i aritmetika pointera su veoma bliski.
- Po definiciji, naziv niza je upravo adresa nultog elementa niza.

  `p = &a[0]` je ekvivalentno sa `p = a`

  `a[i]` se može napisati kao `*(a+i)`
- Ukoliko primenimo operator & na oba ova oblika, dobijamo da je

  `&a[i]` isto što i `a+i`
- Slično važi i za pokazivač p, pa je

  `p[i]` isto što i `*(p+i)`
- Postoji razlika između pokazivača i nizova
  - p=a i p++ je dozvoljeno
  - a=p i a++ nije dozvoljeno

Pokazivači i nizovi

- Kada se funkciji prosleđuje naziv niza, ono što se zapravo šalje je adresa nultog elementa.
- Unutar pozvane funkcije ovaj argument je lokalna promenljiva, a parametar koji predstavlja niz je pokazivač, odnosno promenljiva koja sadrži adresu nultog elementa niza.
- Posmatrajmo funkciju za određivanje dužine stringa (niza karaktera):

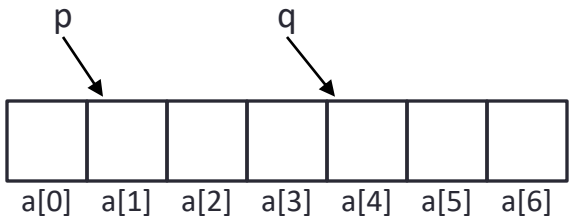
```
/* duzina: vraca duzinu stringa s */
int duzina(char *s)
{
    int n;
    for(n = 0; *s != '\0'; s++)  n++;
    return n;
}
...

duzina("Zdravo, drugari!");      /* konstantan string */
duzina(niz);       /* char niz[100]; */
duzina(pok);       /* char *pok */
```

Pokazivači i nizovi

- U definiciji funkcija dozvoljeno je deklarisanje formalnih parametara na oba načina – i kao niza i kao pokazivača.
- Deklaracije
  `char s[]` i `char *s`
  su ekvivalentne.
- Funkciji je moguće proslediti i deo nekog niza tako što bi joj se prosledio pokazivač na podniz. Tako, ukoliko je `a` niz, pozivi
  `fun(&a[5]);` i `fun(a+5);`
  prosleđuju funkciji `fun` podniz koji počinje od petog elementa niza `a`.
- Ako u nekom nizu postoje, elementi se mogu indeksirati i unazad
  `p[-1], p[-2], ...`
  ukazujući na elemente koji se nalaze neposredno ispred p[0]
- Nije ispravno koristiti objekte koji nisu unutar granica niza.

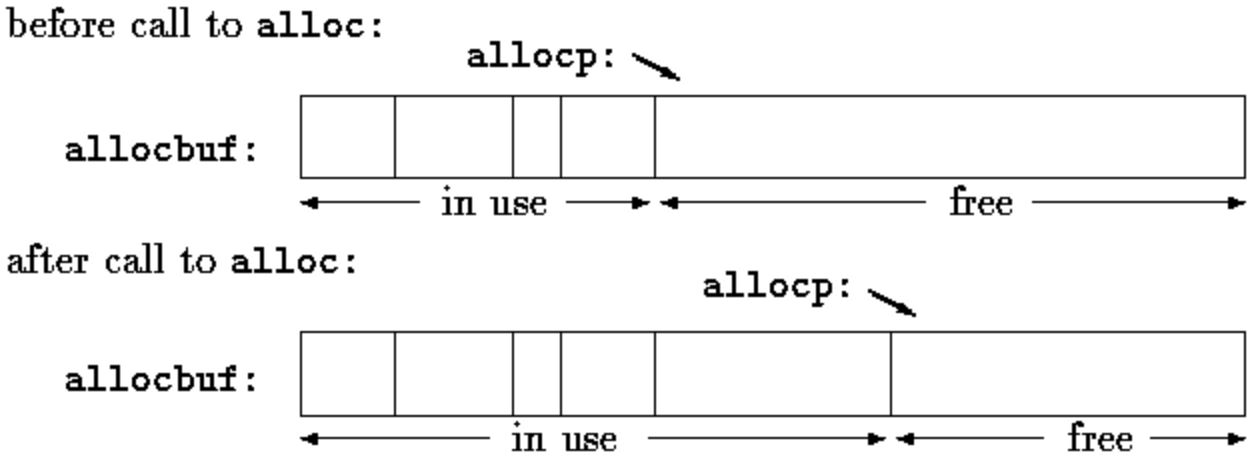
Adresna aritmetika

- Ako je p pokazivač na neki element niza, tada p++ uvećava p tako da pokazuje na naredni element
- p+=i uvećava p tako da pokazuje na element koji se nalazi i elemenata iza onog na koji trenutno p pokazuje
- Ako pokazivači p i q pokazuju na članove istog niza, onda se
  - mogu primeniti operatori ==, !=, <, >, <=, >=
    p < q je tačno ukoliko p pokazuje na element koji je ispred onog na koji pokazuje q
  - q – p predstavlja broj elemenata niza koji se nalaze između

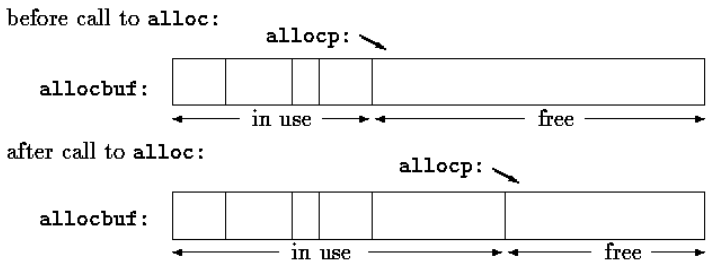


Primer: Funkcije za alokaciju memorije

- `alloc(n)` – obezbeđuje memoriju za n uzastopnih znakovnih pozicija i vraća pokazivač na prvi član tog niza
- `afree(p)` – oslobađa memoriju koja je na ovaj način rezervisana i na koju pokazuje pokazivač p
- funkcije **`afree`** se moraju pozivati u suprotnom redosledu od onog u kojima je pozivana funkcija **`alloc`**

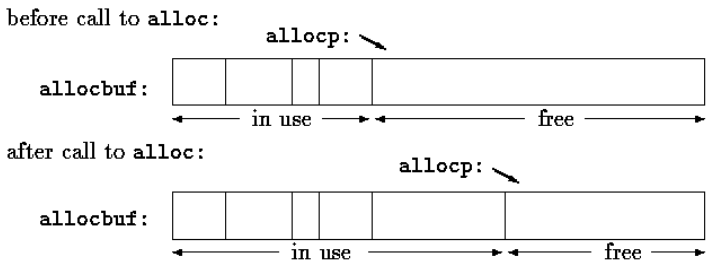


Primer: Funkcije za alokaciju memorije




```
#define ALLOCSIZE 10000 /* size of available space */

static char allocbuf[ALLOCSIZE]; /* storage for alloc */
static char *allocp = allocbuf;  /* next free position */

char *alloc(int n) /* return pointer to n characters */
{
      if (allocbuf + ALLOCSIZE - allocp >= n) /* it fits */
      {
               allocp += n;
               return allocp - n; /* old p */
      }
      else /* not enough room */
               return 0;
}

void afree(char *p) /* free storage pointed to by p */
{
      if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
               allocp = p;
}
```

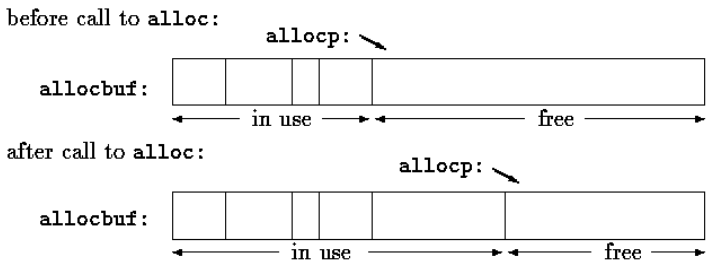
Primer: Funkcije za alokaciju memorije




```
#define ALLOCSIZE 10000 /* size of available space */

static char allocbuf[ALLOCSIZE]; /* storage for alloc */
static char *allocp = allocbuf;  /* next free position */

char *alloc(int n) /* return pointer to n characters */
{
      if (allocbuf + ALLOCSIZE - allocp >= n) /* it fits */
      {
               allocp += n;
               return allocp - n; /* old p */
      }
      else /* not enough room */
               return 0;
}

void afree(char *p) /* free storage pointed to by p */
{
      if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
               allocp = p;
}
```

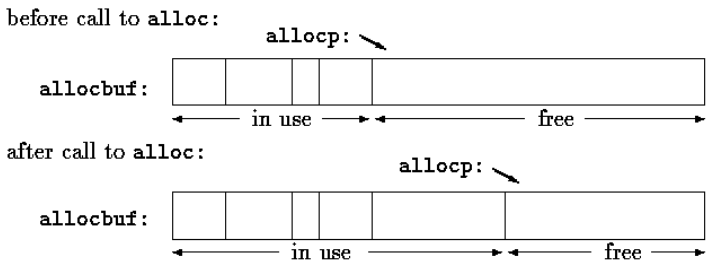
```
static char *allocp;
⋮
allocp = allocbuf;
```

Primer: Funkcije za alokaciju memorije




```
#define ALLOCSIZE 10000 /* size of available space */

static char allocbuf[ALLOCSIZE]; /* storage for alloc */
static char *allocp = allocbuf;  /* next free position */

char *alloc(int n) /* return pointer to n characters */
{
      if (allocbuf + ALLOCSIZE - allocp >= n) /* it fits */
      {
               allocp += n;
               return allocp - n; /* old p */
      }
      else /* not enough room */
               return 0;
}

void afree(char *p) /* free storage pointed to by p */
{
      if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
               allocp = p;
}
```

Primer: Funkcije za alokaciju memorije




```
#define ALLOCSIZE 10000 /* size of available space */

static char allocbuf[ALLOCSIZE]; /* storage for alloc */
static char *allocp = allocbuf;  /* next free position */

char *alloc(int n) /* return pointer to n characters */
{
      if (allocbuf + ALLOCSIZE - allocp >= n) /* it fits */
      {
               allocp += n;
               return allocp - n; /* old p */
      }
      else /* not enough room */
               return 0;
}

void afree(char *p) /* free storage pointed to by p */
{
      if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
               allocp = p;
}
```

ili `return NULL;`

Funkcije za alokaciju i oslobađanje memorije u standardnoj biblioteci

```
#include <stdlib.h>     /* malloc, calloc, realloc, free */

int main ()
{
  int *buffer1, *buffer2, *buffer3;
  buffer1 = (int*) malloc (100*sizeof(int));
  buffer2 = (int*) calloc (100, sizeof(int));
  buffer3 = (int*) realloc (buffer2, 500*sizeof(int));
  free (buffer1);
  free (buffer3);
  return 0;
}
```

- malloc – alokacija dela memorije veličine datog broja bajtova
- calloc – alokacija dela memorije za n objekata navedene veličine i inicijalizuje nulom svaki objekat
- realloc – realokacija
- free – oslobađanje prostora prethodno zauzetog funkcijama malloc, calloc ili realloc

Primer: Funkcija `strlen`

```
/* strlen: return length of string s */
int strlen(char *s)
{
        char *p = s;
        while (*p != '\0')
                p++;
        return p - s;
}
```

- Dozvoljene operacije sa pokazivačima:
  - dodeljivanje pokazivača istog tipa
  - sabiranje i oduzimanje pokazivača i celog broja
  - oduzimanje i poređenje dva pokazivača
  - dodeljivanje nule pokazivaču i njegovo poređenje sa nulom
- Nije dozvoljeno:
  - sabiranje, množenje i deljenje dva pokazivača
  - pomeranje i maskiranje po bitovima
  - dodavanje brojeva tipa float i double pokazivačima
  - dodeljivanje pokazivača jednog tipa pokazivaču drugog tipa bez eksplicitne konverzije (osim za void*)

void pokazivač

- Pokazivač koji nije povezan ni sa jednim tipom podataka
- Može da sadrži adresu bilo kog tipa i može da se konvertuje u bilo koji tip podataka

```
int a = 10;
char b = 'x';

void *p = &a;  // void pointer holds address of int 'a'
p = &b;     // void pointer holds address of char 'b'
```

- Funkcije `malloc()` i `calloc()` vraćaju pokazivače tipa `void` i moraju se konvertovati
- Koriste se za definisanje generičkih funkcija, na primer `comp` ili `swap`

void pokazivač

- void pokazivač se ne može dereferencirati

```
int a = 10;
void *ptr = &a;
printf("%d", *(int *)ptr);
```

- Neki kompajleri ne podržavaju pointersku artimetiku nad void pokazivačima

```
int a[2] = {1, 2};
void *ptr = a;
ptr = ptr + sizeof(int);
printf("%d", *(int *)ptr);
```

Znakovni pokazivači i funkcije

- `"Ja sam string"` – string konstanta

```
printf("hello, world\n");
```

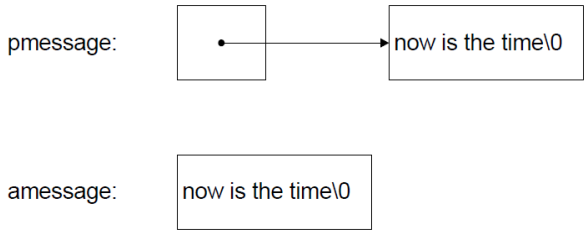
  - `printf` dobija pokazivač na početak niza znakova

```
char *pmessage;
pmessage = "now is the time";
```

- `pmessage` je pokazivač na dati niz znakova; ovom dodelom se ne vrši kopiranje stringova

Znakovni pokazivači i funkcije

```
char amessage[] = "now is the time"; /* niz */
char *pmessage = "now is the time"; /* pokazivač */
```



- `amessage` je niz,
  - veliki tačno onoliko koliko je potrebno za smeštanje navedenog niza znakova;
  - članovi niza mogu menjati vrednosti
- `pmessage` je pokazivač
  - inicijalizovan da pokazuje na string konstntu;
  - njegova vrednost se može promeniti, ali string konstana se ne može modifikovati

- Kopiranje stringa t u string s nije moguće izvršiti komandom `s = t`

Primer: Funkcija `strcpy`

- Funkcija `strcpy(s,t)` kopira string `t` u string `s`
- Verzija sa nizovima

```
/* strcpy: copy t to s; array subscript version */
void strcpy(char *s, char *t)
{
    int i=0;
    while ((s[i] = t[i]) != '\0')   i++;
}
```

- Verzija sa pokazivačima

```
/* strcpy: copy t to s; pointer version */
void strcpy(char *s, char *t)
{
    while ((*s = *t) != '\0')  {
            s++;
            t++;
    }
}
```

Primer: Funkcija `strcpy`

- Iskusni C programeri bi napisali

```
/* strcpy: copy t to s; pointer version 2 */
void strcpy(char *s, char *t)
{
      while ((*s++ = *t++) != '\0');
}
```

- Ili još kraće

```
/* strcpy: copy t to s; pointer version 3 */
void strcpy(char *s, char *t)
{
      while (*s++ = *t++);
}
```

Primer: Funkcija `strcmp`

- Funkcija `strcmp(s,t)` poredi stringove `s` i `t` i vraća negativnu vrednost, nulu ili pozitivnu vrednost u zavisnosti od toga da li je s leksikografski ispred, jednako ili iza t

- Verzija sa nizovima

```
/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
    int i;
    for (i = 0; s[i] == t[i]; i++)
        if (s[i] == '\0')
            return 0;
    return s[i] - t[i];
}
```

Primer: Funkcija `strcmp`

- Verzija sa pokazivačima

```
/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
    for ( ; *s == *t; s++, t++)
        if (*s == '\0')
            return 0;
    return *s - *t;
}
```

Pokazivač na pokazivač

- Učitavanje niza u funkciji

```
void Unos(int *a, int *n)      /* pogresno */
{
     int i;
     scanf("%d",n);
     a = (int*)malloc(*n * sizeof(int));
     for(i=0; i<*n; i++)
          scanf("%d",&a[i]);
}

...
int *a, n;
Unos(a, &n)
...
```

Pokazivač na pokazivač

- Učitavanje niza u funkciji

```
void Unos(int **a, int *n)  /* version 1 */
{
     int i;
     scanf("%d",n);
     *a = (int*)malloc(*n * sizeof(int));
     for(i=0; i<*n; i++)
          scanf("%d",&(*a)[i]);
}

...
int *a, n;
Unos(&a, &n)
...
```

Pokazivač na pokazivač

- Učitavanje niza u funkciji

```
int* Unos(int *n)  /* version 2 */
{
     int i, *a;
     scanf("%d",n);
     a = (int*)malloc(* n *sizeof(int));
     for(i=0; i<*n; i++)
          scanf("%d",&a[i]);
     return a;
}

...
int *a, n;
a = Unos(&n)
...
```