Programiranje 2

Predavanje 06 - pokazivači, stringovi, argumenti komandne linije, pokazivači na funkcije

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

08. travnja 2024.

Broj riječi u stringu

Pišemo funkciju koja broji riječi u stringu koji funkcija prima kao argument. Riječ je niz znakova bez praznina, a riječi su odvojene bar jednom prazninom ili znakom `\t`.

Problem možemo riješiti na dva načina:

- koristimo varijablu `razmak` kojom pamtimo jesmo li pozicionirani ispred riječi (na razmaku), u tom slučaju je postavimo na istina, ili smo unutar riječi kada ju postavimo na laž. Svaka promjena vrijednosti varijable od laž prema istina označava jednu riječ (moramo paziti na zadnju riječ).
- koristimo varijablu `rijec` koju postavimo na istinu kada se pozicioniramo na riječ, a kada dođemo do razmaka nakon riječi, postavimo na laž. Svaka promjena varijable iz laž u istina označava jednu riječ.

Broj riječi u stringu

```
1  int broj_rijeci(char *str){
2  int brojac = 0, razmak = 1;
3
4  while (*str != '\0') {
5        if (*str == '␣' || *str == '\t') {
6              if (!razmak) {/* Izlaz iz rijeci.*/
7                    ++brojac;
8                    razmak = 1; } }
9        else /* Dio rijeci. */
10             razmak = 0;
11       ++str; }
12
13 if (!razmak) ++brojac; /* Zadnja rijec! */
14
15 return brojac;
16 }
```

Broj riječi u stringu

```
1  int broj_rijeci(char *str){
2  int brojac = 0, rijec = 0;
3
4  for ( ; *str != '\0'; ++str)
5        if (*str == '␣' || *str == '\t') {
6            if (rijec)
7                rijec = 0;  }
8       else /* Dio rijeci. */
9           if (!rijec) { /* Ulaz u rijec. */
10               ++brojac;
11               rijec = 1; }
12
13 return brojac;
14 }
```

Broj riječi u stringu

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int broj_rijeci(char *str);
5
6  int main(void) {
7  char s[] = "Ja␣sam␣mala␣Ruza,␣mamina␣sam␣kci.";
8
9  printf("String:\n");
10 printf("%s\n", s);
11 printf("Broj␣rijeci:␣%d\n", broj_rijeci(s));
12
13 return 0;
14 }
```

Izlaz za ulazni string *s* gore:

Broj rijeci: 7

Broj riječi u stringu - varijacije

- Promijenite funkciju `broj_rijeci` tako da separatori budu i znakovi: ' ', '\t', '\v', '\n', '\r', '\f'.
- Dodatno i znakovi: ',', ':', ';', '.', '!', '?'.
- Promijenite funkciju `broj_rijeci` tako da broji samo riječi sastavljene od slova.
- Promijenite funkciju `broj_rijeci` tako da broji samo brojeve sastavljene od dekadskih znamenaka.
- Promijenite funkciju `broj_rijeci` tako da broji samo operatore po pravilima C-a.
- Nalazi najdulju riječ s određenim svojstvom i vraća pokazivač na početak te riječi (ili `NULL` ako je nema).

Implementacija funkcije atoi

Pišemo implementaciju funkcije `atoi` iz standardne biblioteke (zaglavlje `<stdlib.h>`). Ova funkcija pretvara niz znakova (string), koji sadrži zapis cijelog broja, u njegovu numeričku vrijednost. Funkcija treba: preskočiti sve početne bjeline (kao u funkciji `scanf`, i redom pročitati sve znamenke broja, te ih pretvoriti u broj tipa `int`.

```
1  #include <ctype.h>
2
3  int f_atoi(const char s[]) {
4  int i, n, sign; /* Indeks, broj, predznak. */
5
6  /* Preskace sve bjeline, prazan for. */
7  for (i = 0; isspace(s[i]); ++i) ;
8  sign = (s[i] == '-') ? -1 : 1; /* Predznak! */
```

Implementacija funkcije atoi

```
9  /* Preskoci predznak, ako ga ima. */
10 if (s[i] == '+' || s[i] == '-') ++i;
11
12 /* Hornerov algoritam za broj. */
13 for (n = 0; isdigit(s[i]); ++i)
14 n = 10 * n + (s[i] - '0');
15
16 return sign * n;
17 }
```

Koristimo funkcije `isspace` i `isdigit` iz `<ctype.h>`.

Funkcije konverzije iz <stdlib.h>

U standardnoj biblioteci <stdlib.h> postoje neke funkcije konverzije za pretvaranje stringa u broj odgovarajućeg tipa:

```
14 double atof(const char *s) //pretvara s u double,
15 int atoi(const char *s) //pretvara s u int,
16 long atol(const char *s) //pretvara s u long.
```

Dijele puno svojstava s funkcijom za čitanje broja odgovarajućeg tipa:

- preskaču se bjeline na početku stringa
- čita se najdulji niz znakova koji odgovara pravilu za pisanje broja tog tipa
- pročitani niz znakova se pretvara u broj

Funkcije konverzije iz <stdlib.h> - primjer

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void) {
5  printf("%d\n", atoi("␣123")); /* 123 */
6  printf("%d\n", atoi("␣12␣3")); /* 12 */
7  printf("%d\n", atoi("␣12abc3")); /* 12 */
8  printf("%d\n", atoi("abc12␣3")); /* 0 */
9  //ne moze konvertirati abc u vrijednost
10 //tipa int -> tada vraca 0
11 printf("%g\n", atof("␣12.5a4c␣3")); /* 12.5 */
12 printf("%g\n", atof("␣12.5e4c␣3")); /* 125000 */
13 return 0;}
```

U <stdlib.h> su definirane i funkcije za apsolutnu vrijednost `int abs(int n)` i `long labs(long n)`, dok je funkcija `double fabs(double d)` deklarirana u `math.h`.

Ispis cijelog broja u string - itoa

Pišemo implementaciju funkcije itoa koja pretvara cijeli broj u string (slično kao kod ispisa).

```
1  void f_itoa(int n, char s[]){
2  int i = 0, sign; /* Indeks, predznak. */
3  /* Zapamti predznak u sign i
4  pretvori n u nenegativan broj. */
5  if ((sign = n) < 0) n = -n;
6  /* Generiraj znamenke u obratnom poretku. */
7  do {
8        s[i++] = n % 10 + '0'; /* Znamenka. */
9  } while ((n /= 10) > 0); /* Obrisi ju. */
10
11 if (sign < 0)
12        s[i++] = '-'; /* Dodaj minus na kraj. */
13 s[i] = '\0'; /* Kraj stringa. */
```

Ispis cijelog broja u string - itoa

```
14  invertiraj(s);
15
16  return;
17  }
```

Funkcija `f_itoa` ne radi korektno za najmanji prikazivi cijeli broj n. Razmislite zašto, te ispravite kod da ispravno radi za sve prikazive cijele brojeve.

Zadaci - funkcije `atof` i `ftoa`

Napišite implementaciju funkcije `atof` (`<stdlib.h>`) za pretvaranje niza znakova (stringa), koji sadrži zapis realnog broja, u njegovu numeričku vrijednost (tipa `double`). Realni broj smije biti napisan po svim pravilima C-a za pisanje realnih konstanti (točka, e).

Napravite implementaciju funkcije `ftoa` koja pretvara realni broj (tipa `double`) u string (kao kod ispisa u `printf`).

Polja pokazivača

Polje pokazivača ima deklaraciju:

```
tip_pod *ime[izraz];
```

Primarni operator [] ima **viši prioritet** od unarnog operatora *.

```
int *ppi[10]; //polje od 10 pokazivaca
int (*ppi)[10]; //pokazivac na polje od
                //10 elemenata
```

Pokazivač na char možemo inicijalizirati stringom. Isto vrijedi i za polje takvih pokazivača — dobivamo **polje stringova**.

```
static char *gradovi[] = { "Osijek", "Rijeka",
                           "Split", "Zagreb"};
```

Često se koristi za fiksna imena objekata. Npr. dani u tjednu, mjeseci u godini.

Polje pokazivača

Postoji bitna razlika između polja pokazivača (na znak, odnosno, string):

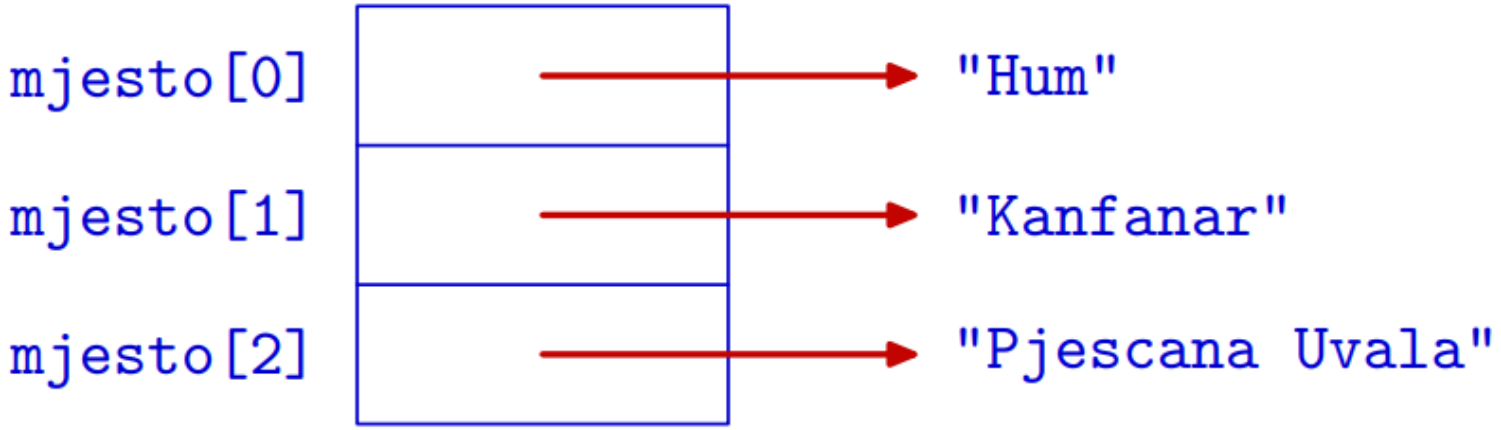
```
char *mjesto[] = { "Hum", "Kanfanar",
                   "Pjescana␣Uvala" };
```

i dvodimenzionalnog polja znakova:

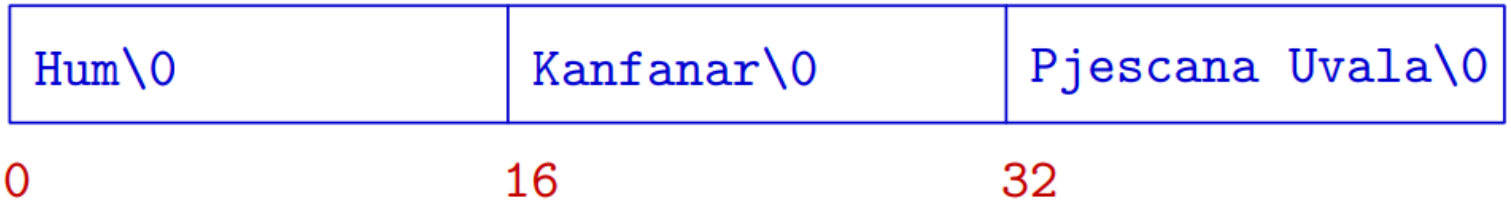
```
char amjesto[][16] = { "Hum", "Kanfanar",
                       "Pjescana␣Uvala" };
```

Polje pokazivača

Polje pokazivača:



Dvodimenzionalno polje znakova:



Sortiranje rječnika

Riječi raznih duljina:

- učitavamo sa standardnog ulaza
- leksikografski sortiramo (uzlazno)
- ispišemo u sortiranom poretku

Pojedine riječi treba spremati kao stringove. Pretpostavit ćemo da svaka pojedina riječ stane u string od 80 znakova (inače koristiti dinamičko alociranje stringova). Također, pretpostavljamo da ulaz sadrži po jednu riječ u svakom redu (ideja, jednostavno čitanje stringova funkcijom `get_s` ili `fgets`).

Sortiranje rječnika

Dva načina spremanja riječi:

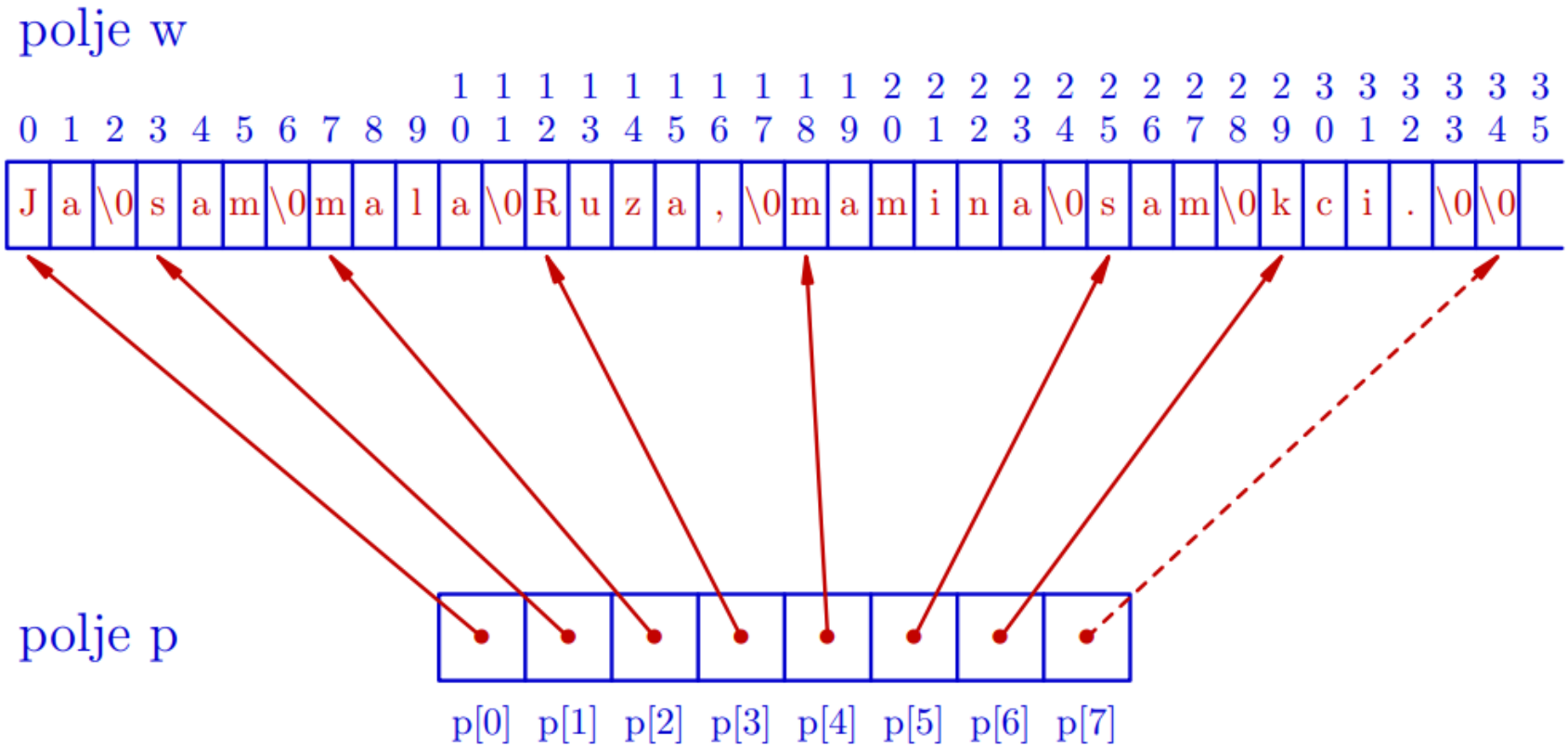
- Dvodimenzionalno polje (sortiramo niz redaka rijeci). Navedeni pristup je loš iz dva razloga: a) spremanje je memorijski rastrošno, b) sortiranje je vrlo sporo zbog zamjena stringova (polja).
- Polje pokazivača na stringove u kojem i-ti element sadrži pokazivač na i-tu riječ. Sortiranje riječi realiziramo zamjenama pokazivača u polju, a ne stringova.

Koristit ćemo dva jednodimenzionalna polja:

- polje znakova w koje sadrži sve riječi (kao stringove), jednu za drugom, bez praznina
- polje pokazivača na znakove p tako da $p[i]$ sadrži pokazivač na početak $i + 1$-e riječi u polju w

Pretpostavit ćemo da polje p sadrži maksimalno 100 riječi.

Sortiranje rječnika



Sortiranje rječnika

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  /* Sortiranje rjecnika. */
6  /* Simbolicke konstante:
7  MAXBROJ = max. broj rijeci,
8  MAXDULJ = max. duljina pojedine rijeci. */
9  #define MAXBROJ 100
10 #define MAXDULJ 80
11
12 /* Globalno polje znakova za rjecnik. */
13 char w[MAXBROJ * MAXDULJ];
14 /* Globalno polje pokazivaca na
15 pojedine rijeci - stringove. */
16 char *p[MAXBROJ];
```

Sortiranje rječnika

```
17 /* Stvarni broj rijeci u rjecniku. */
18 int broj_rijeci;
19
20 /* Ucitava rijeci i vraca broj rijeci.
21 Kraj ucitavanja je prazna rijec. */
22 int unos(char *p[]){
23 char *q = w;
24 int broj = 0, dulj;
25
26 while (1) {
27         if (broj >= MAXBROJ) return -1;
28         p[broj] = fgets(q, MAXDULJ, stdin);
29         if ((dulj = strlen(q)) == 0) break;
30         q += dulj + 1;
31         ++broj; }
32 return broj; }
```

Sortiranje rječnika

```
33 /* Sortiranje rjecnika izborom ekstrema.
34 Dovodimo najmanji element na pocetak.
35 Rjecnik je zadan poljem pokazivaca na rijeci
36 (element polja je pokazivac na znak-string).
37 Koristimo samo zamjene pokazivaca. */
38
39 void sort(char *p[], int n){
40 int i, j, ind_min;
41 char *temp;
42
43 for (i = 0; i < n - 1; ++i) {
44         ind_min = i;
45         for (j = i + 1; j < n; ++j)
46               if (strcmp(p[j], p[ind_min]) < 0)
47                   ind_min = j;
```

Sortiranje rječnika

```
48                  if (i != ind_min) {
49                      temp = p[i];
50                      p[i] = p[ind_min];
51                      p[ind_min] = temp;
52                  }
53              }
54  return;
55  }
56
57  /* Ispisuje sve rijeci u rjecniku. */
58  void ispis(char *p[]){
59  int i;
60
61  for (i = 0; i < broj_rijeci; ++i)
62              puts(p[i]);
63  return; }
```

Sortiranje rječnika

```
/* Glavni program. */
int main(void){

if ((broj_rijeci = unos(p)) >= 0) {
        printf("Broj rijeci = %d\n", broj_rijeci);
        sort(p, broj_rijeci);
        ispis(p); }
else
       printf("Previse rijeci na ulazu.\n");
return 0; }
```

Sortiranje rječnika - varijacije na temu

U prethodnom programu, polja *w* i *p* imaju fiksnu duljinu (zadali smo maksimalni broj riječi i maksimalnu duljinu riječi). Promijenite program tako da se polja *w* i *p* **dinamički alociraju** i po potrebi **realociraju** (ako treba povećati duljinu nekog polja).

Umjesto polja *w*, koristite dinamičku alokaciju za svaku učitanu riječ tako da je ili zadana maksimalna duljina riječi (npr. 80 znakova) ili uz realokaciju riječi u malim blokovima ili *znak-po-znak*.

Sortiranje rječnika - zadaci

- Prilagodite i iskoristite `QuickSort` algoritam za sortiranje rječnika. Pazite na to da funkcija `swap` mora zamijeniti vrijednosti pokazivača na znakove (stringove). Stoga argumenti funkcije swap moraju biti pokazivači na te pokazivače, tip argumenata je `char **`.
- Za sortiranje rječnika iskoristite funkciju `qsort` iz standardne biblioteke `<stdlib.h>`. Treba paziti na tipove, kao za `swap` i koristiti pokazivače na funkcije.

Argumenti komandne linije

Programi vrlo često koriste parametre koji se zadaju zajedno s imenom programa, te se učitavaju iz tog programa. Takvi parametri zovu se argumenti komandne linije:

```
cp ime1 ime2
```

Ako želimo koristiti argumente komandne linije, moramo funkciju `main` deklarirati s dva formalna argumenta (a ne kao do sada s `void`). Standardna imena za te argumente su `argc` tipa `int` i `argv`, polje pokazivača na `char` (polje stringova).

```
1 int main(int argc, char *argv[])
2 { ... }
```

Argumenti komandne linije

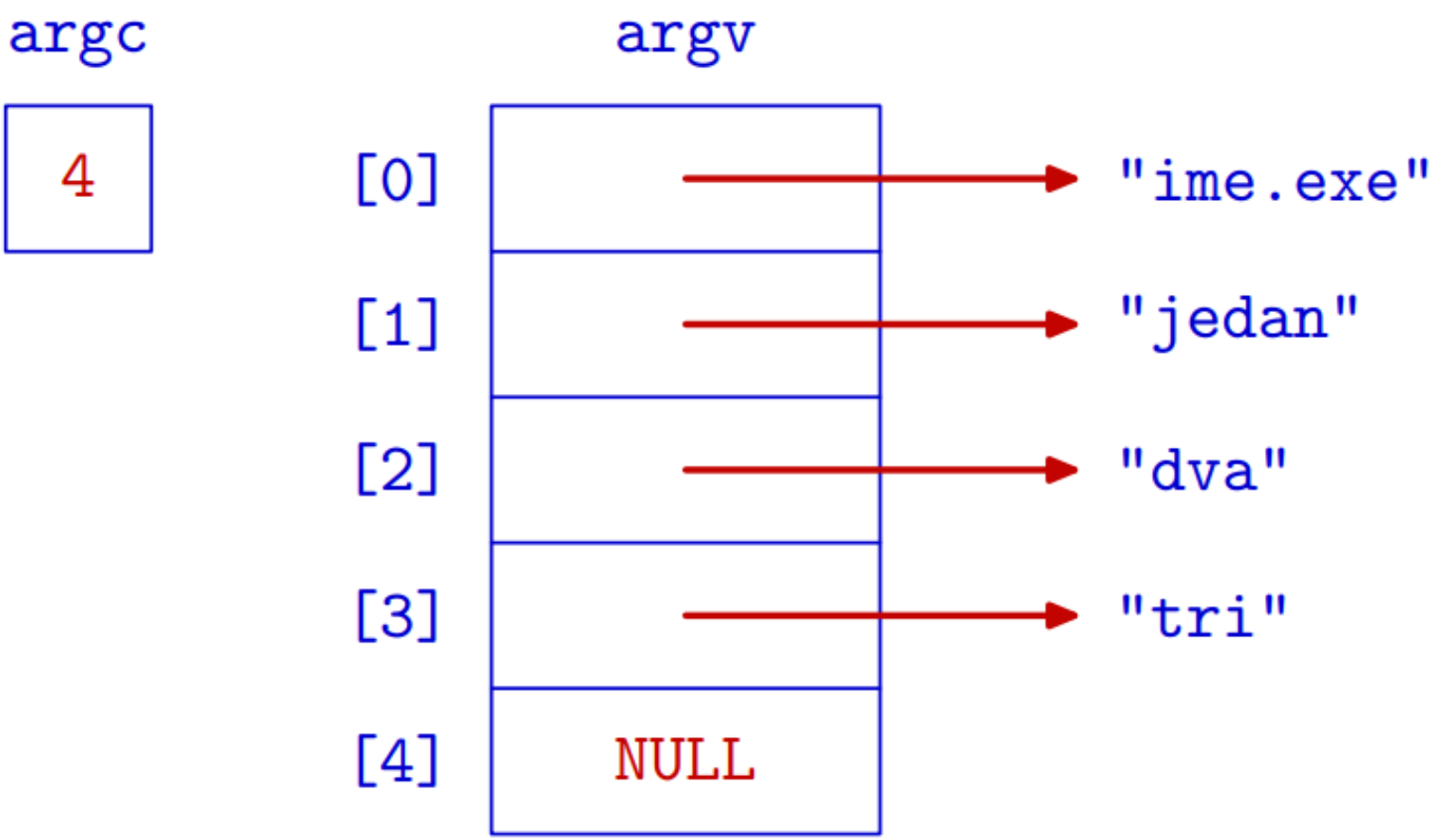
`argc` (*argument count*). Broj `argc` $-$ 1 je broj argumenata komandne linije. Ako nema argumenata komandne linije, `argc` $= 1$.

`argv` (*argument value*) je polje pokazivača na argumente komandne linije. `argv[0]` uvijek pokazuje na string koji sadrži ime programa, kako je pozvan na komandnoj liniji. Ostali parametri smješteni su redom kojim su upisani. Uvijek je `argv[argc] = NULL`. Argumenti se učitavaju kao stringovi u funkciji `scanf`, bjeline su separatori (osim kada je argument u navodnicima).

Argumenti komandne linije

Ako program pozovemo s:

```
ime.exe jedan dva tri
```



Argumenti komandne linije

Program koji ispisuje argumente komandne linije:

```
1  #include <stdio.h> /* Program arg_1. */
2
3  int main(int argc, char *argv[]){
4  int i;
5  printf("argc␣=␣%d\n", argc);
6
7  for (i = 0; i < argc; ++i)
8       printf("argv[%d]:␣%s\n", i, argv[i]);
9
10 return 0;
11 }
```

Čitanje brojeva s komandne linije

Često u programima učitavamo jedan broj (recimo n), nakon čega učitavamo n brojeva, riječi, znakova itd. Ukoliko učitane elemente želimo spremiti, trebamo i alocirati memoriju, gdje opet koristimo zadanu vrijednost varijable n.

Broj n možemo učitati i iz **komandne linije**, ali tada trebamo pretvoriti dobiveni `string` u `int` (koristiti funkciju `atoi`).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(int argc, char *argv[]){
4 ...
5 n = atoi(argv[1]);
6 ...
7 }
```

Provjera argumenata komandne linije

Ako program očekuje neke argumente na komandnoj liniji, onda uvijek treba provjeriti:

- jesu li zaista upisani pri pozivu programa - test `argc`
- imaju li svi `argv[i]` korektan (očekivani) oblik

```
if (argc < 2) {
     printf("Broj elemenata nije zadan.\n");
     exit(EXIT_FAILURE); } /* exit(1); */

n = atoi(argv[1]);
if (n <= 0) {
    printf("Broj elemenata nije pozitivan.\n");
    exit(EXIT_FAILURE); } /* exit(1); */
```

U `CodeBlocks`-u koristiti projekte.

Pokazivač na funkciju

Pokazivač na funkciju deklarira se kao:

```
tip_pod (*ime)(tip_1 arg_1, ..., tip_n arg_n);
```

Ovdje je `ime` varijabla tipa pokazivač na funkciju, koja uzima *n* argumenata, tipa `tip_1` do `tip_n`, i vraća vrijednost tipa `tip_pod`. Slično kao i u prototipu funkcije, u deklaraciji ne treba pisati imena argumenata `arg_1` do `arg_n`.

```
int (*pf)(char c, double a);
int (*pf)(char, double);
```

Zagrade su nužne kod deklaracije pokazivača na funkciju zato što primarni operator () ima viši prioritet od unarnog operatora *.

Pokazivač na funkciju

Treba razlikovati funkciju koja vraća pokazivač na povratnu vrijednost nekog tipa (u primjeru `double`) od pokazivača na funkciju koja vraća element tipa `double`.

```
1 double *pf(double, double);
2 double *(pf(double, double));
3 //funkcije koje vracaju double*
```

```
1 double (*pf)(double, double);
2 //pokazivac na funkciju
3 //koja vraca vrijednost tipa double
```

Pokazivač na funkciju

Pokazivač na funkciju omogućava da jedna funkcija prima neku drugu funkciju kao argument. Realizacija ide tako da prva funkcija dobiva pokazivač na drugu funkciju.

```
1 int prva(int, int (*druga)(int));
```

U pozivu prve funkcije navodimo samo stvarno ime druge funkcije (koja negdje mora biti deklarirana s tim imenom), tj. ime funkcije je sinonim za pokazivač na tu funkciju.

```
1 prva(n, stvarna_druga);
```

Pokazivač na funkciju - primjer

Treba izračunati vrijednost integrala zadane (realne) funkcije f na segmentu $[a, b]$

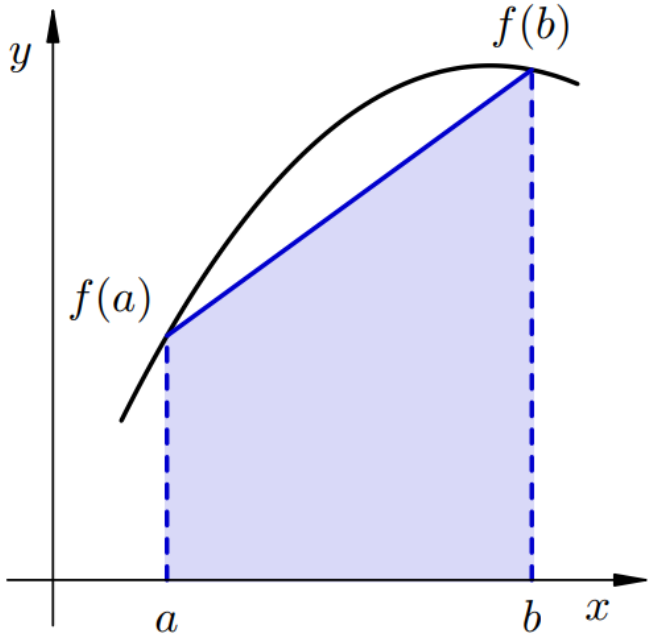
$$I = \int_a^b f(x)dx$$

Za računanje integrala obično se koriste približne (numeričke) formule. Slično Riemannovim sumama, te formule koriste vrijednosti funkcije u određenim točkama iz $[a, b]$. Funkcija za (približno) računanje integrala I onda mora imati (barem) 3 argumenta: granice integracije a,b i funkciju koja računa vrijednost $f(x)$ u zadanoj točki x.

Trapezna formula

Napišimo funkciju koja približno računa integral zadane funkcije, po tzv. trapeznoj formuli. Trapezna formula ima oblik (površina = srednjica puta visina trapeza):

$$\int_a^b f(x)dx \approx \frac{f(a)+f(b)}{2}(b-a)$$



Trapezna formula - program

```
1  include <stdio.h>
2  #include <math.h>
3
4  double integracija(double, double,
5  double (*)(double));
6
7  int main(void) {
8  printf("Sin: %f\n", integracija(0, 1, sin));
9  printf("Cos: %f\n", integracija(0, 1, cos));
10
11 return 0; }
12
13 double integracija(double a, double b,
14 double (*f)(double)) {
15    return 0.5 * (b - a) * ( (*f)(a) + (*f)(b) );
16 }//ova metoda nije jako tocna, provjerite!
```

Produljena trapezna formula

- Izaberemo prirodan broj $n \in \mathbb{N}$
- Segment $[a, b]$ podijelimo na n podintervala točkama x_i, $i = 0, \ldots, n$ tako da je

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$$

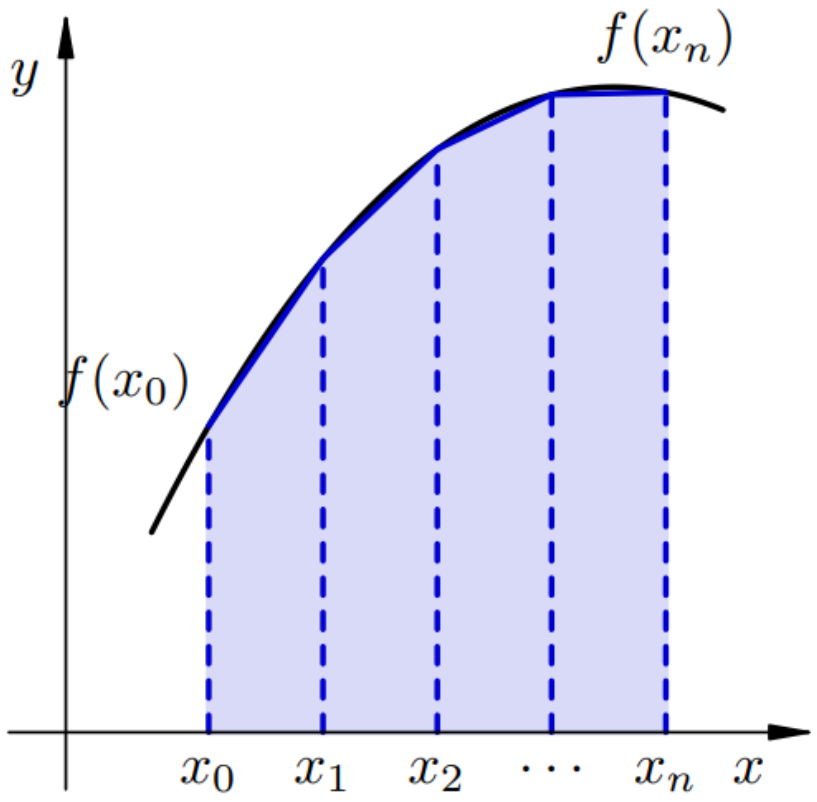
  Pripadni podintervali su $[x_{i-1}, x_i]$ za $i = 1, \ldots n$
- Vrijedi:

$$\int_a^b f(x)dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)dx$$

- Na svakom podintervalu $[x_{i-1}, x_i]$, za $i = 1, \ldots, n$, iskoristimo običnu trapeznu formulu i sve zbrojimo.

Obično se točke x_i uzimaju ekvidistantno, tako da svi podintervali imaju jednaku duljinu (h). Onda je **duljina** podintervala $h = \frac{b-a}{n}$, a točke su $x_i = a + i \cdot h, i = 0, \ldots, n$.

Produljena trapezoidna formula



Običnom trapezoidnom formulom na intervalu $[x_{i-1}, x_i]$ dobivamo:

$$\int_{x_{i-1}}^{x_i} f(x)dx \approx \frac{f(x_{i-1}) + f(x_i)}{2}(x_i - x_{i-1}) = \frac{h}{2}(f(x_{i-1}) + f(x_i))$$

Produljena trapezoidna formula

Zbrajanjem po svim podintervalima dobijemo:

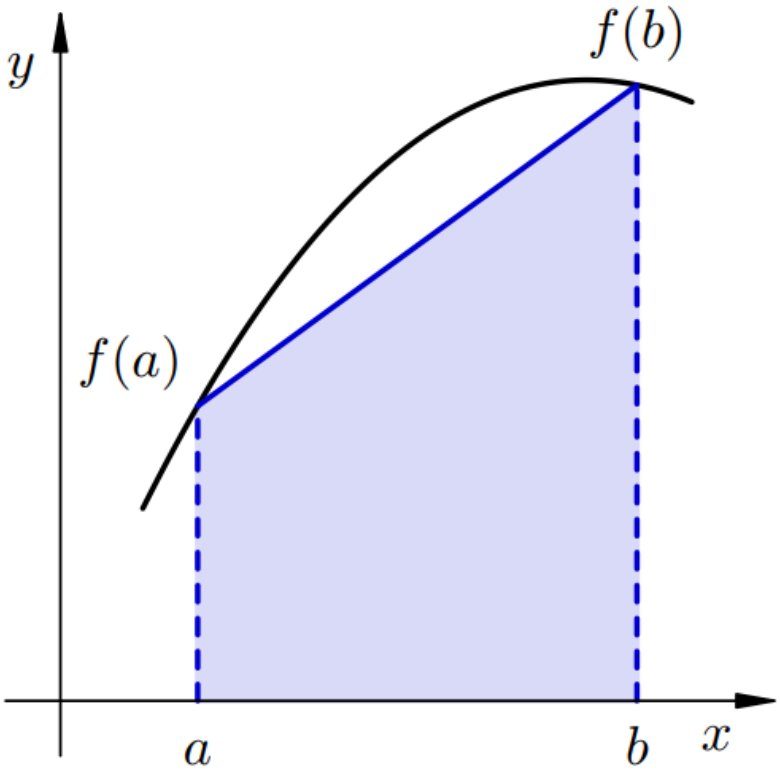
$$I \approx I_n = \sum_{i=1}^{n} \frac{h}{2}(f(x_{i-1}) + f(x_i)) =$$

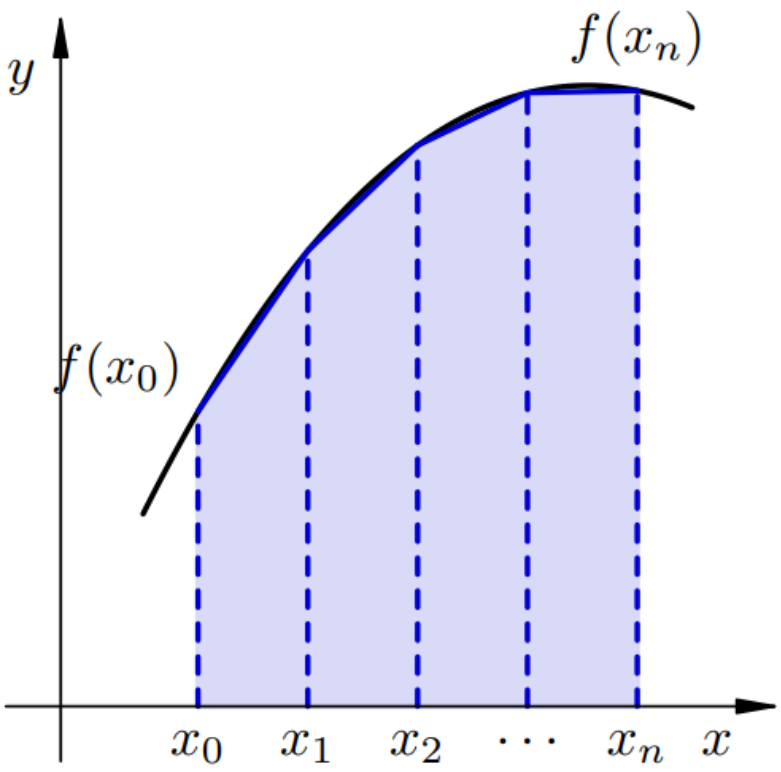
$$\frac{h}{2}(f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n))$$

$$= h(\frac{1}{2}(f(a) + f(b)) + \sum_{i=1}^{n-1} f(a + i \cdot h))$$

Produljena trapezoidna formula

Povećavanjem broja n dobivamo sve točniju aproksimaciju integrala:





Produljena trapezoidna formula

```
1  #include <stdio.h>
2  #include <math.h>
3  double integracija(double, double, int,
4  double (*)(double));
5
6  double integracija(double a, double b, int n,
7  double (*f)(double)){
8
9  double suma, h = (b - a) / n;
10 int i;
11 suma = 0.5 * ( (*f)(a) + (*f)(b) );
12
13 for (i = 1; i < n; ++i)
14         suma += (*f)(a + i * h);
15 return h * suma;
16 }
```

Produljena trapezoidna formula

```
int main(void){

double a = 0.0, b = 2.0 * atan(1.0); /* pi/2 */
int n = 1;
printf("Integral␣sinusa␣od␣0␣do␣pi/2:\n");

while (n <= 100000) {
        printf("␣[n␣=␣%6d]:␣%13.10f\n", n,
        integracija(a, b, n, sin));
        n *= 10;}

return 0;
}
```

Ime funkcije je pokazivač na tu funkciju

Za pravilno deklariranu funkciju, ime funkcije je sinonim za pokazivač na tu funkciju. Zato u pozivu funkcije `integracija` navodimo samo ime funkcije `sin` kao stvarni argument:

```
integral = integracija(0, 1, sin);
```

Adresni operator & ispred `sin` nije potreban (kao ni ispred imena polja), iako ga je dozvoljeno napisati:

```
integral = integracija(0, 1, &sin);
```

Slično smijemo napraviti i kod poziva funkcije zadane pokazivačem. Ne treba dereferencirati taj pokazivač.

```
double integracija(double a, double b,
double (*f)(double)) {
return 0.5 * (b - a) * ( (*f)(a) + (*f)(b) );
}//potpuno korektno
```

Ime funkcije je pokazivač na tu funkciju

```
1 double integracija(double a, double b,
2 double (*f)(double)) {
3 return 0.5 * (b - a) * ( f(a) + f(b) ); }
```

Gornji kod je dozvoljen iako ne sasvim korektan po pitanju tipova. f je pokazivač na funkciju a $*f$ je funkcija nakon dereferenciranja.

Gornji kod možemo zapisati i kao:

```
1 double integracija(double a, double b,
2 double (*pf)(double)) {
3 double (*f)(double) = pf;
4 return 0.5 * (b - a) * ( f(a) + f(b) ); }
```

Pokazivač na funkciju - zadaci

Napišite funkciju koja, kao argumente, prima string (tj. pokazivač na `char`) te pokazivač na funkciju za provjeru znakova, i radi sljedeće:

- vraća broj takvih znakova u stringu
- to isto, a kroz varijabilni argument vraća prvi takav znak u stringu, ako ga ima (u protivnom, ne mijenja taj argument)
- vraća pokazivač na prvi takav znak u stringu, ako ga ima (u protivnom, vraća `NULL`)
- to isto, kroz varijabilni argument vraća broj takvih znakova u stringu
- vraća pokazivač na zadnji takav znak u stringu, ako ga ima (u protivnom, vraća `NULL`).