

---

# Mergesort

---

implementacija pomoću lista

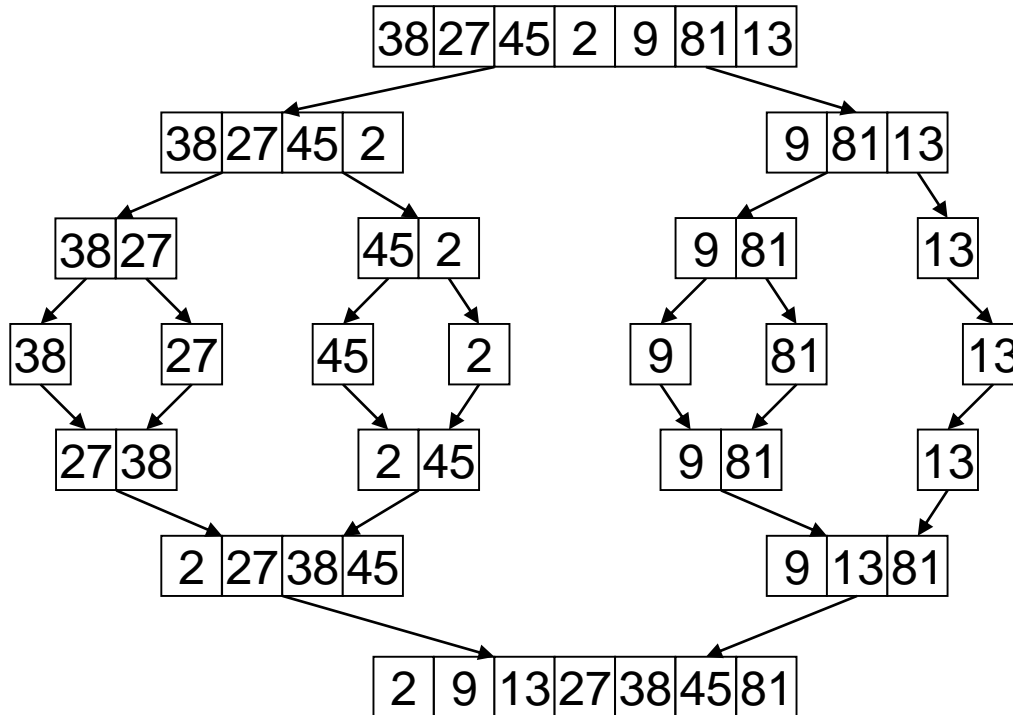
---

# Mergesort

- dijeli nesortirani niz na dva podniza „jednake” duljine
  - sortira, na isti način, svaki od nastalih podnizova
  - spaja (*merge*) dva sortirana podniza u novi sortirani niz.
  
  - Složenost:  $O(n \log_2 n)$
  
  - Autor: John von Neumann, 1945.
-

# Mergesort - implementacija

- Sortiranje niza od 7 elemenata.



---

# Mergesort - implementacija

```
struct list {  
    int number;  
    struct list *next;  
};
```

```
/* Dodavanje elementa polja u listu. */
```

```
struct list *addlist(int number, struct list *next);
```

```
/* Funkcija mergesort na vezanoj listi. */
```

```
struct list *mergesort(struct list *head);
```

```
/* Funkcija merge – spajanje sortiranih lista. */
```

```
struct list *merge(struct list *head_one, struct list *head_two);
```

---

# Mergesort –funkcija `main`

```
int main(void) {
    struct list *head;
    struct list *current;
    struct list *list_ptr;
    int test[] = {38, 27, 45, 2, 9, 81, 13, 7, 9, 0};
    int i;

    head = NULL;
    /* Pretvaranje polja u vezanu listu. */
    for(i = 0; i < 10; i++)
        head = addlist(test[i], head);

    /* Sortiranje liste.*/
    head = mergesort(head);
```

---

# Funkcija main (2)

```
/* Ispis lista. */
```

```
printf(" prije poslije \n"), i = 0;
```

```
for(current = head; current != NULL; current = current -> next)  
    printf("%4d\t%4d\n", test[i++], current -> number);
```

```
/* Oslobadjanje memorije. */
```

```
for(current = head; current != NULL; current = list_ptr)  
    list_ptr = current -> next, free(current);
```

```
return 0;
```

```
}
```

---

# Funkcija `addlist`

```
struct list *addlist(int number, struct list *list_ptr) {  
    struct list *tlist;  
  
    tlist = (struct list*)malloc(sizeof(*tlist));  
  
    if(tlist != NULL) {  
        tlist -> number = number;  
        tlist -> next = list_ptr;  
    }  
  
    return tlist;  
}
```

---

# Funkcija mergesort

```
struct list *mergesort(struct list *head) {
    struct list *head_one;
    struct list *head_two;

    if((head == NULL) || (head -> next == NULL)) return head;

    head_one = head;
    head_two = head -> next;
    while((head_two != NULL) && (head_two -> next != NULL)) {
        head = head -> next;
        head_two = head_two -> next -> next;
    }
    head_two = head -> next;
    head -> next = NULL;
    return merge(mergesort(head_one), mergesort(head_two));
}
```



# Funkcija merge

```
struct list *merge(struct list *head_one, struct list *head_two) {
    struct list *head_three;

    if(head_one == NULL) return head_two;

    if(head_two == NULL) return head_one;

    if(head_one -> number < head_two -> number) {
        head_three = head_one;
        head_three -> next = merge(head_one -> next, head_two);
    } else {
        head_three = head_two;
        head_three -> next = merge(head_one, head_two -> next);
    }
    return head_three;
}
```

---

# Složenost Mergesorta

- Pretpostavka:  $n = 2^k$  ( $k = \log_2 n$ )
- Rekurzivna relacija:

$$\square T(n) = \begin{cases} c_1, n = 1 \\ 2T\left(\frac{n}{2}\right) + c_2n, n > 1 \end{cases}$$

$$\begin{aligned} \square T(n) &= 2T\left(\frac{n}{2}\right) + c_2n = 2\left(2T\left(\frac{n}{4}\right) + c_2\frac{n}{2}\right) + c_2n = 2^2T\left(\frac{n}{2^2}\right) + 2c_2n = \dots \\ &= 2^k T\left(\frac{n}{2^k}\right) + kc_2n = c_1n + n \log_2 n \in O(n \log_2 n). \end{aligned}$$

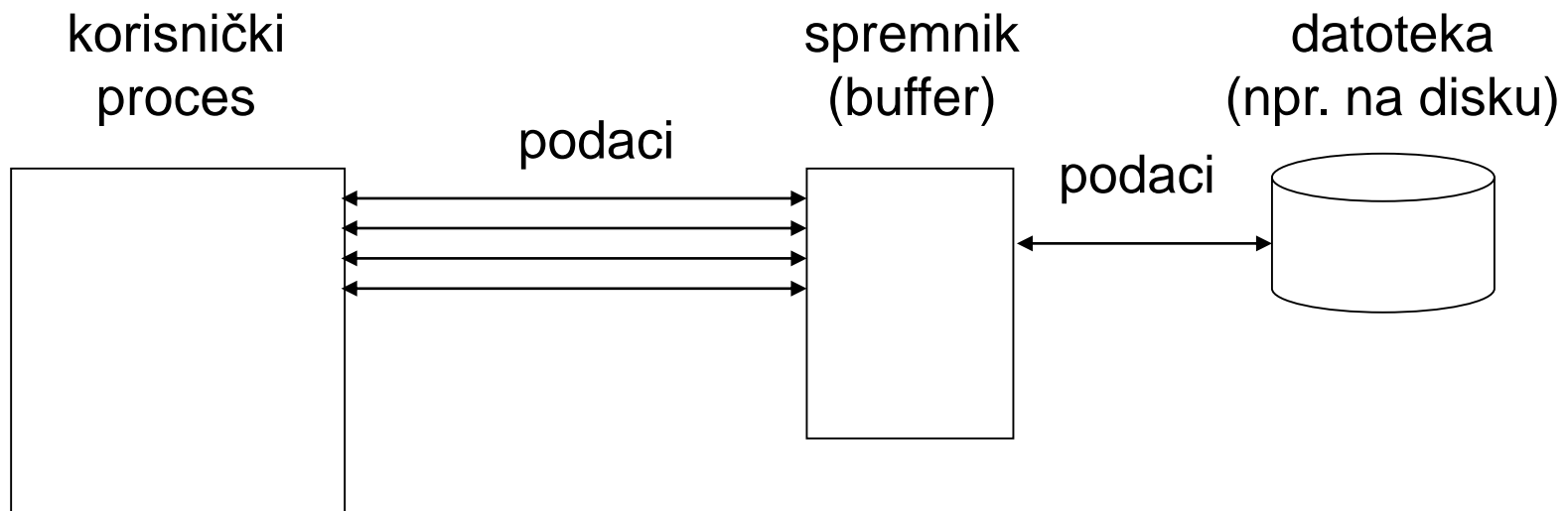
---

# Datoteke

---

# Rad s datotekama

- Međuspremnički način prijenosa podataka (*buffered file system*): komunikacija sa standardnom datotekom vrši se preko spremnika (*buffer*) u koji se privremeno pohranjuju informacije koje se šalju u datoteku.



---

# Rad s datotekama (2)

- Svrha spremnika je smanjiti komunikaciju s vanjskom memorijom (diskom) i tako povećati efikasnost ulazno-izlaznih funkcija.
- Spremnik se kreira deklaracijom pokazivača na strukturu `FILE`:

```
FILE *fp;
```

- `FILE` je posebna struktura definirana u `<stdio.h>` koja definira spremnik. *fp* je pokazivač koji pokazuje na početak spremnika.
  - Datoteka mora biti otvorena prije nego što se može u nju pisati ili iz nje čitati.
-

---

# Podjela datoteka

- Po načinu pristupa:
    - slijedne
    - direktne.
  - Po načinu upisa:
    - formatirane
    - neformatirane.
  - Napomena: Prvo ćemo se upoznati s funkcijama za rad s formatiranim datotekama.
-

---

# Funkcija `fopen`

- Otvaranje datoteke vrši se pomoću funkcije `fopen`. Tipično, `fopen` se koristi na sljedeći način:

```
FILE *fp;
fp = fopen(ime, tip);
if (fp == NULL)
{
    printf(" Greška!\n ");
    .....
}
```

gdje je *ime* ime datoteke (npr. "Moja\_datoteka") koja se otvara, a tip je jedan od sljedećih stringova:

---

# Tipovi (file\_mod):

tip	značenje
"r"	Otvaranje postojeće (tekstualne) datoteke samo za čitanje.
"w"	Kreiranje nove datoteke samo za pisanje.
"a"	Otvaranje postojeće datoteke za dodavanje teksta.
"r+"	Otvaranje postojeće datoteke za čitanje i pisanje.
"w+"	Kreiranje nove datoteke za čitanje i pisanje.
"a+"	Otvaranje postojeće datoteke za čitanje i dodavanje teksta.



---

# Funkcija `fopen` (2)

- Ako se postojeća datoteka otvori s "w" ili s "w+" njen sadržaj će biti izbrisan i pisanje će početi od početka!
  - Ako datoteka koju otvaramo s "a" ili s "a+" ne postoji bit će kreirana, a ako postoji novi tekst će biti dodavan na kraj (**a**ppend) datoteke.
  - Funkcija `fopen` vraća pokazivač na strukturu `FILE` povezanu s datotekom.
  - Funkcija vraća `NULL` u slučaju da datoteka nije mogla biti otvorena.
-

---

# Funkcija `fclose`

- Na kraju programa datoteka treba biti zatvorena funkcijom `fclose` koja kao argument uzima pokazivač na spremnik:

```
fclose(fp);
```

## Primjer:

```
#include <stdio.h>
```

```
FILE *fp;
```

```
if ((fp = fopen("primjer.dat", "w")) == NULL)
```

```
    printf("Nije moguće otvoriti datoteku!\n");
```

```
.....
```

```
fclose(fp);
```

---

---

# Standardne „datoteke”

- Svakom programu stoje na raspolaganju tri automatski otvorene standardne “datoteke”:
    - standardni ulaz (tipkovnica računala)
    - standardni izlaz (ekran računala)
    - standardni izlaz za greške (ekran računala).
  - U datoteci `<stdio.h>` definirani su konstantni pokazivači na `FILE` strukturu povezanu s tim datotekama. Ti pokazivači imaju imena:
    - `stdin`
    - `stdout`
    - `stderr`.
-

---

# Funkcije za čitanje i pisanje

## ■ Funkcije

- `int getc(FILE *fp);`
- `int fgetc(FILE *fp)`

omogućuju učitavanje jednog znaka iz datoteke (međuspremnik) na koju pokazuje `fp`.

- Razlika između tih funkcija je u tome da `getc` može biti implementirana kao makro naredba dok `fgetc` ne smije.
  - U slučaju greške ili kraja datoteke vraća se `EOF`.
  - Funkcija `getchar()` implementira se kao `getc(stdin)`.
-

# Funkcija `ungetc`

## ■ Funkcija

- `int ungetc(int c, FILE *fp);`

vraća znak `c` „natrag” u datoteku na koju pokazuje `fp` i čini ga dostupnim za ponovno čitanje

- U slučaju uspjeha vraća cjelobrojnu vrijednost od `c` a u slučaju neuspjeha `EOF`.

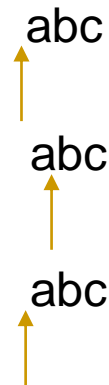
Primjer:

```
char z;
```

```
z = getc(fp);
```

```
ungetc(z, fp);
```

```
z = getc(fp);
```



---

# Primjer:

```
FILE *fp;
```

```
if ((fp = fopen ("kolokvij.txt", "r")) == NULL){  
    fprintf (stderr, "Ne mogu čitati");  
    exit(1);  
}
```

```
while ((c = getc(fp)) != EOF){  
... }
```

```
fclose (fp);
```

---

---

# Funkcije `putc` i `fputc`

- Funkcije

- `int putc(int c, FILE *fp)`

- `int fputc(int c, FILE *fp)`

upisuju znak `c` u datoteku na koju pokazuje `fp` i vraćaju upisani znak.

- Razlika između tih funkcija je u tome da `putc` može biti implementirana kao makro naredba dok `fputc` ne smije.

- U slučaju greške vraća se `EOF`.

- Funkcija `putchar(c)` implementira se kao `putc(c, stdout)`.
-

---

# Primjer:

- Kopiranje sadržaja jedne datoteke u drugu.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[] )
{
    FILE *in, *out;
    int c;

    if (argc != 3) {
        fprintf (stderr, "%s\n", argv[0]);
        exit(1);
    }
}
```

---



# Primjer: (nastavak)

```
if ((in = fopen (argv[1], "r")) == NULL){
    fprintf (stderr, "Ne mogu čitati-%s\n", argv[1]);
    exit(1);
}
if ((out = fopen (argv[2], "w")) == NULL){
    fprintf(stderr, "Ne mogu pisati-%s\n", argv[2]);
    exit(1);
}
while ((c = getc(in)) != EOF)
    putc(c, out);
fclose(in);
fclose(out);
return 0;
}
```

---