ASP – 2. CIRKUS PREDAVANJA – KAMPANJSKI

REKURZIJA

```
int rekurzija (int n...) {
'naredbe koje se obavljaju redom poziva funkcije'
rekurzija;
'naredbe koje se izvode obrnutim redom poziva funkcije'
}
```

!! ako sam dobro shavtio, ako funkcija vraća argument preko naredbe return, onda ona ne može biti tipa void

Primjer za Fibonacijeve brojeve:

```
int Fibo (int n) {
if (n<1) return 1;
else return Fibo (n-2) + Fibo (n-1);
}
```

!! za slučajeve koji se rješavaju rekurzivno, svaki sljedeći rekurzivni poziv mora se približiti osnovnim slučajevima

SORTIRANJE

SORTIRANJE BIRANJEM (SELECTION SORT)

!! pronađi najmanji član niza i zamijeni ga s prvim članom i tako ponavljaj s ostatkom niza smanjujući nesortirani dio

```
void SS (int a [], int n) {
int i, j, min;
for (i=o;i<n;i++) {
min = i;
for (j=i+1;j<n;j++) {
if (a[j]<a[min<]) min = j;
}
Zamijeni (&a[i], &a[min]);
}
}
```

BUBBLE SORT

!! zamijeni susjedne elemente ako je prvi veći od drugog, kreni od početka i gledaj dva po dva, ponavljaj dok niz nije sortiran

```
void BS (int a [], int n) {
int i, j;
for (i = 0; i < n-1; i++) {
for (j = 0; j < n-1-i; j++) {
```

```
if (A[j+1] < A[j])
Zamijeni (&A[j], &A[j+1]);
}
}
}
```

!! s dodatnim uvjetom je li bilo zamjene:

```
void BubbleSort (int A [], int N) {
int i, j, BilaZamjena;
for (i = 0, BilaZamjena = 1; BilaZamjena; i++) {
BilaZamjena = 0;
for (j = 0; j < N-1-i; j++) {
if (A[j+1] < A[j]) {
Zamijeni (&A[j], &A[j+1]);
BilaZamjena = 1;
}
}
}
}
```

SORTIRANJE UMETANJEM (INSERTION SORT)

!! uzemo prvi član koji je veći od svog susjednog desno i zamijenimo ih. Sad nam je to sortirani niz. Dalje uzimamo član i gledamo na koje mjesto bi možda došao taj član u našem već sortiranom dijelu te ga tu stavimo ili je on možda veći od svih pa ga ostavljamo gdje je i idemo dalje.

```
void InsertionSort (int A [], int N) {
int i, j;
int pom;
for (i = 1; i < N; i++) {
pom = A[i];
for (j = i; j >= 1 && A[j-1] > pom; j--)
A[j] = A[j-1];
A[j] = pom;
}
}
```

SHELLOV SORT

!! ovisno o “koraku” gledamo i uspoređujemo članove. Ako naš niz ima 10 brojeva, a korak nam je 3 onda prvo provjeravamo 1. i 4. član niza (1+**3**=4) te ako je došlo do zamijene gledamo možemo li otići i unazad za taj isti korak pa provjeravamo opet ta dva člana.

```
void ShellSort (int A [], int N) {
int i, j, korak, pom;
for (korak = N / 2; korak > 0; korak /= 2) {
for (i = korak; i < N; i++) {
pom = A [i];
for (j = i; j >= korak && A[j-korak] > pom; j -= korak) {
A [j] = A [j - korak];
```

```
}
A [j] = pom;
}
}
}
```

MERGESORT

!! niz dijelimo na dva dijela pa onda svaki taj novi ponovno na nova dva dijela i tako sve dok ne dobijemo niz od jednog elementa. Zatim se vračamo u koracima tako da opet stvaramo jedan veliki, ali sada sortirani niz. Za primjer kako se to radi kad se vračamo. Ako imamo sad dva podniza od po tri elementa: 246 i 135, uspoređujemo prve elemente 2 i 1, odabiremo manji (znaci 1). Zatim imamo 246 i 35. Sad gledamo 2 i 3, pa nam ostaje 46 i 35 itd. itd. [Ovaj sort se izvodi rekurzivno]

```
void MSort(int A [], int PomPolje[], int lijevo, int desno) {
int sredina;
if (lijevo < desno) {
sredina = lijevo + (desno - lijevo) / 2;
MSort (A, PomPolje, lijevo, sredina);
MSort (A, PomPolje, sredina + 1, desno);
Merge (A, PomPolje, lijevo, sredina + 1, desno);
}
}
```

QUICKSORT

!! Ovisno o tome kako izabiremo stožere. Meni najlaksa metoda je ako uzmemo 3 stožera (prvi, u sredini i zadnji; ako je paran broj članova niza, onda se za onoga u sredini uzima prvi s lijeva; ovo se još valjda i naziva medijan). Usporedimo ta tri člana i razvrstamo ih po mjestima na kojima jesu, ali tako da je skroz desno (znaci, na početku niza) najmanji član, zatim u sredini sljedeći po veličini i zadnji je najveći od ta tri. Sad srednjeg (stožernog) zamijenimo s predzadnjim članom. Jedan kursor zatim kreće s lijeve strane i kreće se sve dok ne dođe do člana većeg od stožernog. S desne strane kreće kursor i kreće se dok ne naiđe na člana manjeg od stožernog. Zatim se ta dva člana zamjenjuju. Kad se ti kursori zaobiđu, stožer se mijenja sa članom na kojem se trenutačno nalazi kursor koji je krenuo s lijeve strane.

!! Kada kao stožer odabiremo samo prvi ili zadnji član, ne mijenjamo im pozicije na početku, a dalje radimo isto, s lijeve strane tražimo član veći od stožera, s desne član manji te ih mijenjamo. Kad se mimoiđu kursori, stožer zamijenimo s mjestom na kojem se nalazi lijevi kursor ako nam je stožer bio zadnji broj, ili desni ako nam je kao stožer uzet prvi broj (ja to logički gledam gdje mi dođe da ne uništi sve). Sada, lijevo od stožera gledamo kao jedan niz i za njega uzimamo novi stožer (zadnji, ili prvi broj) te ga sortiramo, pa zatim desni niz.

!!!! Provježbajte dva tri takva niza i gledajte u rješenja!!!!

STOG

!! mišljenja sam da će nam za ovaj ispit biti dovoljnje ove tri operacije: dodaj, brisi i inicijaliziraj stog.

INICIJALIZACIJA:

```
#define MAXSTOG neki broj
typedef struct {
        int vrh, polje[MAXSTOG];
} Stog;
void init_stog(Stog *stog){
        stog->vrh = -1;          'mislim da se ovaj red može zamijeniti sa:  *vrh=-1;
}
```

DODAVANJE:

```
int dodaj (int element, Stog *stog) {
if (stog->vrh >= MAXSTOG-1) return 0;
stog->vrh++;
stog->polje[stog->vrh] = element;
return 1;
}
```

ili meni draža:

```
int dodaj (int stavka, Stog *stog) {
if (*vrh >= n-1) return 0;
 (*vrh)++;
stog[*vrh] = stavka;
return 1;
}
```

BRISANJE:

```
int skini (int *element, Stog *stog) {
if (stog->vrh < 0) return 0;
*element = stog->polje[stog->vrh];
stog->vrh--;
return 1;
```

ili opet meni draža:

```
int skini (int *stavka, Stog *stog) {
if (*vrh < 0) return 0;
*stavka = Stog[*vrh];
(*vrh)--;
return 1;
}
```

LISTE

!! realizirane poljem, a elementi se nazivaju ATOMI. Svaki se element sastoji od podatkovnog dijela i pokazivača na sljedeći element. Lista radi kao stog, „glava“ liste je kao vrh kod stoga.

DODAVANJE:

```
int dodaj (tip element, Stog *stog) {
```

```
atom *novi;
if ((novi = (atom*) malloc(sizeof(atom))) != NULL) {
novi->element = element;
novi->sljed = stog->vrh;
stog->vrh = novi;
return 1;
}
else return 0;
}
```

BRISANJE:

```
int skini (tip *element, Stog *stog) {
atom *pom;
if (stog->vrh == NULL) return 0;
*element = stog->vrh->element;
pom = stog->vrh->sljed;    /* adresa novog vrha */
free(stog->vrh);            /* obriši stari vrh */
stog->vrh = pom;          /* postavi novi vrh */
return 1;
}
```

!! glupost

REDOVI

!! ono što prvo stavimo u red, to prvo i uzimamo. FIFO. Isto kao kod stoga, inicijalizacija, dodavanje i brisanje. Samo proučite funkcije jednim pogledom, napišete na komadić papira i na kraju vas smjeste u prvu klupu.

INICIJALIZACIJA:

```
typedef struct {
        tip polje[MAXRED];
        int ulaz, izlaz;
} Red;
```

DODAVANJE

```
int dodaj (tip element, Red *red) {
if ((red->ulaz+1) % n == red->izlaz) return 0;
red->ulaz++;
red->ulaz %= n;
red->polje[red->ulaz] = element;
return 1;
}
```

BRISANJE

```
int skini (tip *element, Red *red) {
if (red->ulaz == red->izlaz) return 0;
red->izlaz++;
red->izlaz %= n;
*element = red->polje[red->izlaz];
return 1;
}
```

SRETNO SVIMA!!!!