LISTE

Zavrsni 2014.

Zadatak 1. (6 bodova)

U jednostruko povezanu nesortiranu listu spremljeni su podaci o studentima. Lista je zadana sljedećim strukturama:

```
struct zapis {
  char imeprezime[80+1];
  int ocjena;
};
```

```
struct at {
  struct zapis element;
  struct at *sljed;
};
typedef struct at atom;
```

Napišite funkciju čiji je prototip:

```
void podijeliListu(atom **glava, int ocjena);
```

koja će listu podijeliti s obzirom na zadanu ocjenu i pri tome obrnuti redoslijed elemenata.
Znači, nakon završetka funkcije svi čvorovi sa zapisima o studentima čija je ocjena manja ili jednaka zadanoj ocjeni trebaju biti u prvom dijelu liste u obrnutom redoslijedu od redoslijeda u zadanoj listi, dok ostali čvorovi trebaju biti u drugom dijelu liste, također u obrnutom redoslijedu.

Primjer: prije poziva funkcije lista sadrži sljedeće ocjene: 5, 2, 4, 3, 4, 3, 2, 4, 5, 1.

Nakon poziva funkcije `podijeliListu` sa zadanom ocjenom 3, lista treba biti: 1, 2, 3, 3, 2, 5, 4, 4, 4, 5.

Zadatak 1. (6 bodova)

```
void podijeliListu(atom **glava, int element) {

	atom *lijevo = NULL, *desno = NULL, *sljedeci, *pom = *glava;

	while (pom != NULL) {
		sljedeci = pom->sljed;
		if (pom->element.ocjena <= element) {
			pom->sljed = lijevo;
			lijevo = pom;
		} else {
			pom->sljed = desno;
			desno = pom;
		}
		pom = sljedeci;
	}
	//spoji lijevu i desnu
	if (lijevo == NULL) {
		*glava = desno;
	} else {
		*glava = lijevo;
		while (lijevo->sljed != NULL)
			lijevo = lijevo->sljed;
		lijevo->sljed = desno;
	}
}
```

Zadatak 2. (15 bodova)

U jednostruko povezanu nesortiranu listu spremljeni su podaci o studentima. Lista je zadana sljedećim strukturama:

```
struct zapis {
    char imeprezime[80+1];
    int visina;
};
```

```
struct at {
    struct zapis element;
    struct at *sljed;
};
```

Napišite funkciju čiji je prototip:

```
int rasporediElemente(atom** glava);
```

koja će elemente liste (studente) rasporediti ovisno o odnosu njihove visine i visine studenta koji je u početnoj listi na početku (početni student). U novoj listi, oni studenti koji su viši od početnog studenta trebaju se naći s njegove „lijeve" strane (bliže početku) u obrnutom redosljedu od početnog, dok se ostali studenti trebaju naći s njegove „desne" strane u originalnom redosljedu. Funkcija vraća novu poziciju elementa koji je u početnoj listi bio na početku (na poziciji 1).

Npr. ako su visine studenata u početnoj listi **169**, 196, 160, 161, 171, 197, 158, nakon raspoređivanja trebaju biti 197, 171, 196, **169**, 160, 161, 158 (početak liste je na studentu čija je visina 197). Funkcija treba vratiti 4.

Zadatak 2. (15 bodova)

```
int rasporedi(atom **glava){
  atom *prviVeci,*zadnjiManji, *trenutni, *slijedeci;
  int novaPozicija;
  double vrijednost;

  if(!(*glava)) return 0;

  vrijednost=(*glava)->element.visina; // visina početnog studenta
  novaPozicija=1;
  prviVeci=zadnjiManji=*glava; // pamtim prvog većeg i zadnjeg manjeg
  trenutni=(*glava)->sljed;
  zadnjiManji->sljed=NULL;  // zadnji manji pokazuje na kraj
  while(trenutni){ // dok ima elemenata u listi
    slijedeci=trenutni->sljed; // zapamtim slijedećeg
    if(trenutni->element.visina > vrijednost){ // ako je viši od početnog
      trenutni->sljed=prviVeci; // stavim ga na početak liste
      prviVeci=trenutni;
      novaPozicija++; // i zapamtim novu poziciju
     }
    else{
      zadnjiManji->sljed=trenutni;  // inače ga stavim na kraj
      zadnjiManji= zadnjiManji->sljed;
      zadnjiManji->sljed=NULL;
    }
    trenutni=slijedeci; // prijeđem na novi element liste
  }
  *glava=prviVeci; // vratim novi početak
  return novaPozicija; // i novu poziciju starog početnog…
}
```

Zadatak 3. (15 bodova)

Zadana je dvostruko povezana lista atoma definiranih odsječkom:

```
typedef struct at {
        int vrijednost;
        struct at *slijed;
        struct at *preth;
} atom;
```

Atomi su u listi sortirani uzlazno prema vrijednosti: prvi atom (čiji je pokazivač preth jednak NULL) ima najmanju vrijednost. Dopunite definiciju funkcije brisi koja iterira po listi od prvog elementa prema zadnjem tražeći atom zadane vrijednosti. Ako nađe takav atom, vadi ga iz liste i vraća njegovu adresu, a ako ne nađe takav atom vraća NULL.

```
atom * brisi(atom **glava, int vrijednost) {
  int prvaiteracija = 1;
  atom *pom;
  while(*glava) {
    if(_____> vrijednost) {
      return NULL;
    }
    if(_____ == vrijednost ) {
      if(_____ != NULL) {
        _____ = _____;
      }
      if_____ != NULL) {
        _____ = _____;
      }
      pom = *glava;
      if(prvaiteracija) {
        _____ = _____;
      }
      return pom;
    }
    prvaiteracija = 0;
    _____ = _____;
  }
  return NULL;
}
```

Zadatak 3. (15 bodova)

```
atom * brisi(atom **glava, int vrijednost) {
  int prvaiteracija = 1;
  atom *pom;
  while(*glava) {
    if((*glava)->vrijednost > vrijednost) {
      returnNULL;
    }
    if((*glava)->vrijednost == vrijednost ) {
      if((*glava)->preth != NULL) {
        (* glava)->preth->sljed =  (*glava)->sljed;
      }
      if((*glava)->sljed != NULL) {
        (*glava)->sljed->preth = (*glava)->preth;
      }
      pom = *glava;
      if(prvaiteracija) {
        *glava = (*glava)->sljed;
      }
      return pom;
    }
    prvaiteracija = 0;
    glava = &(*glava)->sljed;
  }
  return NULL;
}
```

Zadatak 2. (14 bodova)

Jednostruko povezana lista sastoji se od atoma definiranih odsječkom:

```
typedef struct st_element {
       int vrijednost;
       struct st_element *sljed;
} element;
```

Napisati funkciju za dodavanje novog podatka (atoma) u listu tako da se očuva sortiranost liste. Funkcija prima pokazivač na pokazivač na korijen liste i podatak tipa `int`, a vraća 0 ili 1 ovisno o uspješnosti umetanja u listu.

Zadatak 2. (14)

```
int dodaj_element( element **korijen, int vrijednost ) {
       element *novi = NULL;
       if( ! ( novi = ( element * ) malloc( sizeof( element ) ) ) ) {
              return 0;
       }
       novi -> vrijednost = vrijednost;
       novi -> sljed = NULL;
       while( * korijen && ( ( * korijen ) -> vrijednost < vrijednost ) ) {
              korijen = & ( * korijen ) -> sljed;
       }
       novi -> sljed = * korijen;
       * korijen = novi;
       return 1;
}
```

Zadatak 2. (6 bodova)

U memoriji postoji jednostruko povezana sortirana lista ostvarena (realizirana) strukturom:

```
struct el {
      struct el* sljedeci;
      int vrijednost;
};
typedef struct el element;
```

Napišite funkciju koja će iz liste ukloniti sva ponavljanja nekog elementa ostavljajući u listi samo **prvi** takav element gledano od početka liste. Memoriju koju su zauzimali uklonjeni elementi treba osloboditi. Funkcija prima pokazivač na prvi član liste, a prototip joj je:

```
void izbaci(element* glava);
```

Zadatak 2. (6 bodova)

```
void izbaci(element *glava){
    element *prvi, *drugi, *preth;
    for(prvi = glava; prvi; prvi = prvi->sljedeci){
        drugi = prvi->sljedeci;
        preth = prvi;
        while(drugi){
            if(drugi->vrijednost == prvi->vrijednost){
                preth->sljedeci = drugi->sljedeci;
                free(drugi);
                drugi = preth->sljedeci;
            }else{
                break;
            }
        }
    }
}
```

2. zadatak (13 bodova)

Zadana je jednostruko povezana lista sljedećom strukturom:

```
typedef struct at{
        int element;
        struct at *sljed;
} atom;
```

Napisati rekurzivnu funkciju koja će promjenom pokazivača na sljedeće elemente obrnuti listu. Nerekurzivno rješenje neće se priznavati.

2. (13 bodova)

```
void obrniListu(atom** glava) {
        atom* prvi;
        atom* pom;

        if (*glava == NULL) return;

        prvi = *glava;
        pom = prvi->sljed;

        if (pom == NULL) return;

        obrniListu(&pom);

        prvi->sljed->sljed = prvi;
        prvi->sljed = NULL;

        *glava = pom;
}
```

4. zadatak (14 bodova)

Napisati funkciju koja će iz jednostruko povezane linearne liste izbrisati sve čvorove koji imaju neparnu vrijednost. Prototip funkcije je **`void brisi (atom **glava)`**. Funkcija ne smije koristiti niti jednu pomoćnu varijablu. **Nije potrebno** oslobađati zauzetu memoriju.

Atom liste zadan je strukturom:

```
typedef struct atom {
  int elem;
  struct atom *sljed;
} atom;
```

4. (14 bodova)

```
void brisi(atom **glava){
        while((*glava)){
                if((*glava)->elem % 2){
                        (*glava) = (*glava)->sljed;
                }else{
                        glava = &(*glava)->sljed;
                }
        }
}
```

3. zadatak (10 bodova)

Napisati funkciju koja će od elemenata dviju jednostruko povezanih lista `prva` i `druga` načiniti listu `t`
uzimajući naizmjence elemente iz lista `prva` i `druga`. Listu `treca` funkcija mora vratiti u pozivajući p
Ako element nema para u drugoj listi, funkcija završava s radom.

Atom liste zadan je strukturom:

```
typedef struct atom {
  int elem;
  struct atom *sljed;
} atom;
```

Primjer 1	Primjer 2	Primjer 3
Prva = {1, 3, 5}	Prva = {1,3}	Prva = {1,3,5}
Druga = {2,4,6}	Druga = {2,4,6}	Druga = {2,4}
Treca = {1,2,3,4,5,6}	Treca = {1,2,3,4}	Treca = {1,2,3,4}

Zad 3. (10 bodova)

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

typedef struct atom {
	int elem;
	struct atom *sljed;
} atom;

atom *spoji(atom *glava1, atom *glava2){
	atom *glava = NULL, *pom = NULL, *pom1 = NULL, *pom2 = NULL;
	//Ako je jedna od lista prazna - vrati NULL
	if(glava1 == NULL || glava2 == NULL){
		return NULL;
	}

	//Niti jedna lista nije prazna
	pom1 = glava1;
	pom2 = glava2;
	//Prvi element trece liste je prvi element iz prve liste
	glava = pom = pom1;
	//Pomakni pomocni pokazivac u prvoj listi na sljedeci element
	pom1 = pom1->sljed;
	//Drugi element trece liste je prvi element iz druge liste
	pom->sljed = pom2;
	//Pomakni pomocni pokazivac u drugoj listi na sljedeci element
	pom2 = pom2->sljed;
	//Pomakni pomocni pokazivac u trecoj listi na sljedeci element
	pom = pom->sljed;

	while(pom1 != NULL && pom2 != NULL){
		//Uzimamo element iz prve liste
		pom->sljed = pom1;
		//Pomakni pomocni pokazivac u prvoj listi na sljedeci element
		pom1 = pom1->sljed;
		//Pomakni pomocni pokazivac u trecoj listi na sljedeci element
		pom = pom->sljed;

		//Uzimamo element iz druge liste
		pom->sljed = pom2;
		//Pomakni pomocni pokazivac u drugoj listi na sljedeci element
		pom2 = pom2->sljed;
		//Pomakni pomocni pokazivac u trecoj listi na sljedeci element
		pom = pom->sljed;
	};

	//Postavljamo sljed pokazivač zadnjeg elementa na NULL
	pom->sljed = NULL;
	return glava;
}
```

4. zadatak (16 bodova)

Napisati funkciju koja u pozivajući program vraća jednostruko povezanu listu u kojoj je pohranjena sekvenca poteza koji rješavaju problem Hanojskih tornjeva za **n** diskova, jedan potez po elementu liste. Atom liste zadan je strukturom:

```
typedef struct atom {
	int n;
	char t1;
	char t2;
	struct atom *sljed;
} atom;
```

gdje je **n** redni broj diska, **t1** oznaka tornja s kojeg se disk uzima a **t2** oznaka tornja na koji se disk stavlja.

Zad 4. (16 bodova)

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

typedef struct atom {
	int n;
	char t1;
	char t2;
	struct atom *sljed;
} atom;

void hanoi(char izvor, char odrediste, char pomocni, int n, atom **glava){
	atom *novi, *p;
	if(n > 0){
		//Rekurzivni poziv
		hanoi(izvor, pomocni, odrediste, n-1, glava);
		//Stvaramo novi element
		novi = (atom*)malloc(sizeof(atom));
		novi->sljed = NULL;
		novi->n = n;
		novi->t1 = izvor;
		novi->t2 = odrediste;
		if(*glava == NULL){
			//Lista je prazna, novi element je ujedno i glava liste
			*glava = novi;
		}else{
			//Novi element dodajemo na kraj liste
			for(p = *glava; p->sljed != NULL; p=p->sljed);
			p->sljed = novi;
		}
		//Rekurzivni poziv
		hanoi(pomocni, odrediste, izvor, n-1, glava);
	}
}
```