Zadaci za vježbu – Liste i redovi

1. U red realiziran jednostruko povezanom listom spremaju se podaci cjelobrojnog tipa (**int**). Napisati funkcije za stavljanje u red i skidanje iz reda. Funkcije trebaju vratiti 1 ako je operacija uspješno obavljena, a 0 inače.

Napisati dodatnu funkciju koja će sve elemente zadanog reda slučajno rasporediti u dva nova reda. Svi redovi trebaju biti realizirani jednostruko povezanom listom. Prilikom realizacije dodatne funkcije nije dopušteno izravno pristupati elementima redova, već treba koristiti ranije napisane funkcije za stavljanje u red i skidanje iz reda. Nove, slučajno napunjene redove, treba vratiti u glavni program.

```
typedef struct z{
    int element;
    struct z *sljed;
}zapis;

int stavi(zapis **ulaz, zapis **izlaz, int e) {
    zapis *novi;
    novi = (zapis *)malloc(sizeof(zapis));
    if (novi == NULL) return 0;

    novi->element = e;

    novi->sljed = NULL;
    if (*izlaz == NULL) *izlaz = novi;
    else (*ulaz)->sljed = novi;

    *ulaz = novi;
    return 1;
}

int skini(zapis **ulaz, zapis **izlaz, int *e) {
    zapis *pom;

    if (*izlaz == NULL) return 0;

    pom = *izlaz;
    *e = pom->element;
    *izlaz = pom->sljed;
    if (*izlaz == NULL) *ulaz = NULL;
    free(pom);

    return 1;
}

void fun3(zapis **u1, zapis **i1, zapis **u2, zapis **i2, zapis **u3, zapis
**i3) {
    int e;

    srand((unsigned int)time(NULL));

    while (skini(u1, i1, &e)) {
        if (rand()%2) stavi(u2, i2, e);
        else stavi(u3, i3, e);
    }
}
```

2. U red realiziran **dvostruko** povezanom listom spremaju se cjelobrojni podaci (**long**). Napisati funkciju za skidanje jednog podatka iz reda i funkciju za stavljanje novog podatka u red. Napisati glavni program u kojem će se, koristeći prije napisanu funkciju, u red staviti brojevi od 1 do 10.

```
typedef struct z {
    long el;
    struct z *sljed;
    struct z *preth;
} zapis;

int stavi(zapis **ulaz, zapis **izlaz, int elm) {

    zapis *novi = (zapis *)malloc(sizeof(zapis));
    if (!novi) return 0;

    novi->el = elm;
    novi->sljed = NULL;

    if (*ulaz == NULL) {
        *ulaz = *izlaz = novi;
        novi->preth = NULL;
    }
    else {
        (*ulaz)->sljed = novi;
        novi->preth = *ulaz;
        *ulaz = novi;
    }

    return 1;
}

int skini(zapis **ulaz, zapis **izlaz, int *elm) {

    zapis * pom;
    if (*ulaz == NULL) return 0;  // može i *izlaz == NULL

    pom = *izlaz;
    *izlaz = (*izlaz)->sljed;
    if (*izlaz == NULL) *ulaz = NULL;
    else (*izlaz)->preth = NULL;

    *elm = pom->el;
    free(pom);
    return 1;

}

void main() {
    zapis *ulaz, *izlaz;
    long i;

    ulaz = izlaz = NULL;
    for (i=1; i<=10; i++) {
        stavi(&ulaz, &izlaz, i);
    }
}
```

3. U jednostruko povezanu listu spremaju se podaci o jelima na meniju u nekom restoranu: šifra jela (**int**), naziv jela (**50+1 znak**), cijena (**float**) i opis jela (**300+1 znak**). Lista je sortirana uzlazno, prema cijeni jela. Napisati funkciju koja će u listu dodati podatke o novom jelu (lista mora ostati sortirana). Funkcija treba imati prototip:

```
int dodaj(zapis **glava, int sif, char naz[], float cijena, char opis[]);
```

```
typedef struct el {
    int sifra;
    char naziv[50+1];
    float cijena;
    char opis[200+1];
} element;

typedef struct z {
    element el;
    struct z *sljed;
} zapis;

int dodaj(zapis **glava, int sif, char naz[], float cijena, char opis[]) {
    zapis *novi;

    novi = (zapis *)malloc(sizeof(zapis));
    if (novi == NULL) return 0;
    novi->el.sifra = sif;
    strcpy(novi->el.naziv, naz);
    novi->el.cijena = cijena;
    strcpy(novi->el.opis, opis);

    while (*glava && (*glava)->el.cijena < cijena) {
        glava = &((*glava)->sljed);
    }

    novi->sljed = *glava;
    *glava = novi;

    return 1;
}
```

4. U dvostruko povezanu listu spremaju se cjelobrojni podaci (**long**). Napisati funkciju koja će imati prototip:

```
int izbaciN(zapis **glava, zapis **rep, int N)
```

i koja će iz liste izbaciti prvih **N** elemenata. Ako u listi ima manje od **N** elemenata, funkcija ih treba izbaciti sve. Funkcija treba vratiti stvarni broj izbačenih elemenata.

```
int izbaciN(zapis **glava, zapis **rep, int N) {
    int i = 0;
    zapis *pom;
    while (*glava && i<N) {
        pom = *glava;
        *glava = pom->sljed;
        free(pom);
        i++;
    }

    if (*glava == NULL) *rep = NULL;
    else (*glava)->preth = NULL;

    return i;
}
```

5. Napisati **rekurzivnu** funkciju koja će **sve** elemente iz reda realiziranog dvostruko povezanom listom prebaciti u objektno realizirani stog na način da nakon njenog izvršavanja prvi element u redu (onaj koji se prvi dobije funkcijom skini) bude prvi element na stogu (onaj koji se prvi skida sa stoga, tj. koji se prvi dobije funkcijom skini). Funkcija mora imati prototip:

```
Stog *RedListomUOOPStog(atom2* glava, atom2* rep)
```

Obavezno koristiti sljedeće definicije:

```
struct at2 {
        int element;
        struct at2 *sljed;
        struct at2 *preth;
};
typedef struct at2 atom2;

// dodavanje u red realiziran dvostruko povezanom listom
// funkcija vraca 1 ako uspije, inace 0
int DodajURed (int element, atom2 **glavap, atom2 **repp);

int SkiniIzReda (int *element, atom2 **glavap, atom2 **repp);

void IspisiRed (atom2 *glava);

int BrisiIzReda (atom2 **glavap, atom2 **repp, int element);


class Stog{
private:
        struct at {
                int element;
                struct at *sljed;
        };
        typedef struct at atom;
```

```
        atom *glava;

public:
        Stog():glava(NULL){}
        void Obrisi();
        void Ispisi();
        int Stavi(int e);
        int Skini(int *e);
        ~Stog(){
                Obrisi();
        }
};


Stog *RedListomUOOPStog(atom2* glava, atom2* rep)
{
     if(!glava)
     {
          Stog *stog = new Stog();
          return stog;
     }
     Stog* stog = RedListomUOOPStog((glava)->sljed, rep);
     stog->Stavi(glava->element);
     return stog;
}
```

OOP LISTA--OBICNO STABLO

Glavni.cpp

```
#include <stdio.h>
#include "ListaOOP.h"
#include "ObicnoStablo.h"
// U zadatku treba napisati funkciju koja prabacuje stablo zadano pokazivačima
// u OOP listu
// Funkcija preko parametara uzima sve što joj je potrebno i mora biti rekurzivna.
// U glavnom programu napuniti stablo proizvoljnim elementima i raditi ispise

void StabloUListu(cvor* stablo, Lista* lista)
{
	if(stablo)
	{
		StabloUListu(stablo->lijevo, lista);
		//ovo punjenje je malo čudno
		//uzima se ascii kod prvog znaka niza (zato jer u listu idu integeri
		//nije mi se radila nova lista sa stringovima :$
		podatak p;
		p.kljuc = p.element = (int)stablo->element[0];
		lista->DodajNaKraj(p);
		StabloUListu(stablo->desno, lista);
	}
}

int main()
{
	cvor* stablo=NULL;
	Lista* lista = new Lista();

	//Funkcija za dodavanje u sortirano stablo
	stablo=upis(stablo, "Pero");
	stablo=upis(stablo, "Marko");
	stablo=upis(stablo, "Ivo");
	stablo=upis(stablo, "Jakov");
	stablo=upis(stablo, "Stjepan");
	//ispis stabla
	ispisinld(stablo);
	printf("\n\n");

	StabloUListu(stablo, lista);
	//ispis liste
	printf("Ispis liste:\n");
	lista->IspisiListu();

	//sad je pravo vrijeme da se obriše lista iz memorije
	delete lista;

	//Stablo je sortirano, tj. odmah pri upisivanju elemenata isti su postali sortirani
	//Ako se u rekurziji odabere inorder obilazak stabla (lijevo-desno) elementi iz sortiranog
	//stabla (lijevo manji, desno veći) će biti uzlazno sortirani u rezultantnoj listi.
```

ObicnoStablo.h

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

struct cv {
  char element[15];
  struct cv *lijevo;
  struct cv *desno;
};
typedef struct cv cvor;

// upisuje u stablo podatke: lijevo manji, desno veci
cvor *upis (cvor *glava, char element[]) {
        int smjer; // odluka o podstablu
        if (glava == NULL) { // prazno (pod)stablo
                glava = (cvor *) malloc (sizeof (cvor));
                if (glava) {
                        strcpy (glava->element, element);
                        glava->lijevo = glava->desno = NULL;
                } else {
                        printf ("U memoriji mena mjesta za upisati '%s'\n", element);
                }
        } else if ((smjer = strcmp (element, glava->element)) < 0) {
                glava->lijevo = upis (glava->lijevo, element);
        } else if (smjer > 0) {
                glava->desno  = upis (glava->desno, element);
        } else {
                printf ("Podatak '%s' vec postoji!\n", element);
        }
        return glava; // pokazivac na zadnji element
}
// obilazak inorder lijevo-desno
void ispisinld (cvor *glava) {
        if (glava != NULL) {
                ispisinld (glava->lijevo);
                printf ("%s \n", glava->element);
                ispisinld (glava->desno);
        }
}
// obilazak inorder desno-lijevo
void ispisindl (cvor *glava) {
        if (glava != NULL) {
                ispisindl (glava->desno);
                printf ("%s \n", glava->element);
                ispisindl (glava->lijevo);
        }
}
```

```
// obilazak preorder
void ispispre (cvor *glava) {
	if (glava != NULL) {
		printf ("%s \n", glava->element);
		ispispre (glava->lijevo);
		ispispre (glava->desno);
	}
}
// obilazak postorder
void ispispost (cvor *glava) {
	if (glava != NULL) {
		ispispost (glava->lijevo);
		ispispost (glava->desno);
		printf ("%s \n", glava->element);
	}
}
// ispis stabla
void ispissta (cvor *glava, int nivo) {
	int i;
	if (glava != NULL) {
		ispissta (glava->desno, nivo+1);
		for (i = 0; i < nivo; i++) printf("    ");
		printf ("%s \n", glava->element);
		ispissta (glava->lijevo, nivo+1);
	}
}
// trazenje cvora u binarnom stablu
cvor *trazi (cvor *glava, char element[]) {
	int smjer;
	if (glava) {
		if ((smjer = strcmp (element, glava->element)) < 0) {
			return trazi (glava->lijevo, element);
		} else if (smjer > 0) {
			return trazi (glava->desno, element);
		}
	}
	return glava; // ili je pronadjen ili NULL;
}
```

//ListaOOP

```
#include <iostream>
#include <ctime>

using namespace std;

struct podatak {
	int kljuc;
	int element;
};
```

```
class Lista {
private:
	struct atom {
		podatak _stavka;
		atom *_sljed;
	};
	atom *_glava;
	atom *_rep;

public:
	Lista();
	Lista(const Lista &izvor);
	~Lista();

	void DodajNaPocetak(podatak stavka);
	void DodajNaKraj(podatak stavka);
	bool DodajSortirano(podatak stavka,bool uzlazno);
	bool NadjiPoKljucu(int kljuc,podatak &stavka);
	bool BrisiSPocetka(podatak &stavka);
	bool BrisiZadanog(int kljuc,podatak &stavka);
	void BrisiSve();
	void IspisiListu();
};


Lista::Lista() {
	_glava = NULL;
	_rep = NULL;
}

Lista::Lista(const Lista &izvor) {
	atom *pAtomPred,*pAtomIzvor;
	if ((izvor._glava==NULL) || (izvor._rep==NULL))
		_glava=_rep=NULL;
	else
	{
		_glava=new atom; // procisceni copy constructor
		_glava->_stavka=izvor._glava->_stavka;
		pAtomPred=_glava;
		for (pAtomIzvor=izvor._glava->_sljed; pAtomIzvor!=NULL;
pAtomIzvor=pAtomIzvor->_sljed)
		{
			pAtomPred->_sljed=new atom;
			pAtomPred=pAtomPred->_sljed;
			pAtomPred->_stavka=pAtomIzvor->_stavka;
		}
		pAtomPred->_sljed=NULL;
		_rep=pAtomPred;
	}
}
```

```
Lista::~Lista() {
	BrisiSve ();
}

void Lista::DodajNaPocetak(podatak stavka) {
	atom *novi = new atom;

	novi->_stavka = stavka;
	novi->_sljed = _glava;
	_glava = novi;
	if (_rep == NULL)
		_rep = novi;
	//cout << " na adresu " << novi;
}

void Lista::DodajNaKraj(podatak stavka) {
	atom *novi = new atom;

	novi->_stavka = stavka;
	novi->_sljed = NULL;
	if (_rep != NULL)
		_rep->_sljed = novi;
	else
		_glava = novi;
	_rep = novi;
	//cout << " na adresu " << novi;
}

bool Lista::DodajSortirano(podatak stavka, bool uzlazno) {
	atom *novi = new atom;
	atom *p;

	novi->_stavka = stavka;
	if (_glava == NULL || (uzlazno ^ (_glava->_stavka.element < stavka.element))) {
		// Dodavanje na pocetak liste
		// koristi se "ekskluzivno ili" smjera i usporedbe
		DodajNaPocetak (stavka);
	} else {
		// Dodavanje iza postojeceg elementa kad:
		// nema sljedećeg ili element u sljedećem je (uzlazno ^ veci od novoga)
		for (p = _glava; (p->_sljed != NULL) &&
			(uzlazno ^ (p->_sljed->_stavka.element > stavka.element)); p = p-
>_sljed);
		// da li takav vec postoji
		if ((p->_sljed != NULL) && (p->_sljed->_stavka.element == stavka.element))
			return 0;
		novi->_sljed = p->_sljed;
		p->_sljed = novi;
		cout << " na adresu " << novi;
	}
	return 1;
}
```

```
bool Lista::NadjiPoKljucu(int kljuc, podatak& stavka) {
	atom *p;

	for (p = _glava; (p != NULL) && (p->_stavka.kljuc != kljuc); p = p->_sljed);
	if (p != NULL) {
		stavka = p->_stavka;
		return true;
	} else
		return false;
}

bool Lista::BrisiSPocetka(podatak& stavka) {
	if (_glava != NULL) {
		stavka = _glava->_stavka;
		_glava = _glava->_sljed;
		// ako je bio jedini
		if (_glava == NULL)
			_rep = NULL;
		return true;
	}
	return false;
}

bool Lista::BrisiZadanog(int kljuc, podatak& stavka) {
#if 0
	atom **glavap, *p;
	glavap = &_glava;
	for (; *glavap && (*glavap)->_stavka.kljuc != kljuc; glavap = &(*glavap)->_sljed);
	if (*glavap) {
		p = *glavap;
		*glavap = (*glavap)->_sljed;
		stavka = p->_stavka;
		if (p->_sljed == NULL )
			_rep = p;
		free (p);
		return true;
	} else
		return false;
#else
	atom *p, *preth = NULL;
	for (p = _glava; (p != NULL); p = p->_sljed){
		if (p->_stavka.kljuc == kljuc){
			if (_glava == p){  // brisanje 1. elementa - promjena glave
				_glava = p->_sljed;
			}else {   // brisanje elementa u tijelu liste
				preth->_sljed = p->_sljed;
			}
			if (_rep == p){  // da li se briše zadnji element?
				_rep = preth;
			}
```

```
                        stavka = p->_stavka;
                        free(p);
                        return true;
                }
                preth = p;
        }
        return false;
#endif
}
void Lista::IspisiListu() {
        atom *pom;
        pom = _glava;
        if (pom == NULL)
                cout <<"Lista je prazna\n";
        else
                for (pom; pom !=NULL; pom = pom->_sljed)
                        cout << "Kljuc=" << pom->_stavka.kljuc << " element="<< pom-
>_stavka.element << endl;
}

void Lista::BrisiSve() {
        atom *pom;
        while (_glava!=NULL)
        {
                pom = _glava;
                _glava = _glava->_sljed;
                free (pom);
        }
        _rep = NULL;
}
```

OOPStablo-ObicnaLista

Glavni.cpp

```
#include <stdio.h>
#include "StabloOOP.h"
#include "ObicnaLista.h"
//U zadatku treba napisati glavni program koji treba stvoriti listu realiziranu pokazivačima
//čiji atom sadrži znakovne nizove od 15 elemenata.  Nju treba napuniti s
//proizvoljnih 5 elemenata. Nakon toga napisati funkciju koja preko parametara prima
//navedenu listu i prepisuje je u stablo te novostvoreno stablo vraća preko imena (return)
//U glavnom programu stablo ispisati inorder postupkom.

Stablo* ListaUStablo(atom* lista)
{
	atom* pomocni;
	Stablo* zaVratiti = new Stablo();
	for(pomocni = lista; pomocni != NULL; pomocni = pomocni->sljed)
	{
		zaVratiti->Dodaj(pomocni->element);
	}
	return zaVratiti;
	//NE raditi delete zaVratiti!!! Memorija mora ostati zauzeta i nakon izvršavanja funkcije
}

//Rekurzivna verzija
Stablo* ListaUStablo2(atom* lista)
{
	//početni uvijet - jedino mjesto gdje se
	//stvara objekt stabla
	if(lista == NULL)
	{
		Stablo* st = new Stablo();
		return st;
	}
	//prima se pokazivač na stablo iz rekurzivinih poziva
	//koje je u svakom pozivu isti!!! (tj. stablo je jedno jedino
	//pa uvijek ima istu adresu)
	Stablo* st = ListaUStablo(lista->sljed);
	st->Dodaj(lista->element);
	//vraćanje pokazivača na višu razinu
	return st;
}
int main()
{
	atom* lista=NULL;
	Stablo* st;
	//Funkcija za dodavanje u listu odmah i sortira!!!
	if(!dodaj(&lista, "Pero")) return 0;
	if(!dodaj(&lista, "Ana")) return 0;
	if(!dodaj(&lista, "Marko")) return 0;
	if(!dodaj(&lista, "Ivo")) return 0;
	if(!dodaj(&lista, "Zdravko")) return 0;
```

```
	//ispis liste
	ispisi(lista);
	printf("\n\n");

	st = ListaUStablo2(lista);
	//ispis stabla inorder postupkom
	printf("Inorder ispis stabla:\n");
	st->Inorder();

	//sad je pravo vrijeme da se obriše stablo iz memorije
	//više jadno malo stabalce nikome ne treba ;)
	delete st;
	//Primijetiti - dobiveno je koso stablo!!! To je inače loše, ali pošto
	//sam već isprogramirao neću smišljati novi primjer
	//Dakle, loše se sortirane elemente dodavati u sortirano stablo - dobije se
	//koso stablo, a mi bi ipak nešto što je blisko potpunome stablu jer onda
	//brže pretražujemo
}
```

ObicnaLista.h

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

struct at {
  char element[15];
  struct at *sljed;
};
typedef struct at atom;

int dodaj (atom **glavap, char* element) {
  atom *novi, *p;
  if ((novi = (atom *) malloc(sizeof(atom))) == NULL)
		return 0;
  strcpy(novi->element, element);
  if (*glavap == NULL || strcmp((*glavap)->element, element) >= 0 ){
    // Dodavanje na pocetak liste
    novi->sljed = *glavap;
    *glavap = novi;
  } else {
	// Dodavanje iza postojeceg elementa kad:
	// a) postojeći atom nema sljedećeg
	// b) element u sljedećem cvoru je veći ili jednak novome
	for (p = *glavap; p->sljed && (strcmp((p->sljed)->element, element) < 0); p = p-
>sljed);
		novi->sljed = p->sljed;
		p->sljed = novi;
  }
  return 1;
}
```

```
// ispis elemenata liste
void ispisi (atom *glava) {
  atom *p=NULL;
  printf ("Ispis elemenata liste:\n",      p->element);
  for (p = glava; p != NULL; p = p->sljed) {
    printf ("%s\n",   p->element);
  }
}
```

StabloOOP.h

```
#include <iostream>
#include <ctime>
using namespace std;

class Stablo {
private:
  struct cvor {
    char element[15];
    cvor *lijevo;
    cvor *desno;
  };
  cvor *_glava;

  void Dodaj(cvor **cv, char *element);
  bool Trazi(cvor *cv, char *element);
  void Inorder(cvor *cv);
  void OslobodiMemoriju(cvor **cv);
public:
  Stablo();
  ~Stablo();
  void Dodaj(char *element);
  bool Trazi(char *element);
  void Inorder();
};

Stablo::Stablo()
{
  _glava = NULL;
}
Stablo::~Stablo()
{
  OslobodiMemoriju(&_glava);
}
void Stablo::OslobodiMemoriju(cvor **cv)
{
  if (*cv != NULL)
  {
    if ((*cv)->lijevo != NULL) OslobodiMemoriju(&(*cv)->lijevo);
    if ((*cv)->desno != NULL) OslobodiMemoriju(&(*cv)->desno);
    free (*cv);
  }
}
```

```
void Stablo::Dodaj(char* element) {
  Dodaj(&_glava, element);
}
void Stablo::Dodaj(cvor **cv, char* element) {
  if (*cv == NULL) {
    *cv = (cvor *) malloc (sizeof (cvor));
    if (*cv == NULL) throw "Nema dovoljno memorije!";
    else
    {
      strcpy((*cv)->element, element);
      (*cv)->lijevo = NULL;
      (*cv)->desno = NULL;
    }
  }
  else
  {
    if (strcmp(element, (*cv)->element) <= 0)
      Dodaj(&(*cv)->lijevo, element);
    else
      Dodaj(&(*cv)->desno, element);
  }
}
bool Stablo::Trazi(char* element) {
  return Trazi(_glava, element);
}
bool Stablo::Trazi(cvor *cv, char *element)
{
  if (cv == NULL) return false;
  else
  {
    if (strcmp(element, cv->element) < 0)
      return Trazi(cv->lijevo, element);
    else if (strcmp(element, cv->element) > 0)
      return Trazi(cv->desno, element);
  }
  return true;
}
void Stablo::Inorder()
{
  Inorder(_glava);
}
void Stablo::Inorder(cvor *cv)
{
  if (cv != NULL) {
    Inorder(cv->lijevo);
    printf("%s \n", cv->element);
    Inorder(cv->desno);
  }
}
```

ZADACI ZA VJEŽBU - STABLA

1. U binarno stablo spremaju se cjelobrojni podaci (**int**). Napisati funkciju koja će pronaći i vratiti koliko se elemenata stabla nalazi na zadanoj razini. Funkcija treba raditi efikasno (ne smije pretraživati čitavo stablo već samo do zadane razine) i treba imati prototip:

```
int funkcija(cvor *korijen, int razina);
```

Napomena: korijen stabla nalazi se na razini 1.

```
typedef struct c {
    int broj;
    struct c *l;
    struct c *d;
}cvor;

int funkcija(cvor *korijen, int razina) {
    if (korijen == NULL) return 0;
    if (razina < 1) return 0;
    if (razina == 1) return 1;
    return funkcija(korijen->l, razina-1) + funkcija(korijen->d, razina-1);
}
```

2. U binarno stablo spremaju se cjelobrojni podaci (**long**). Stablo je sortirano (lijevo manji, desno veći)

a) Napisati **NEREKURZIVNU** funkciju koja će pronaći zadani element u stablu. Ako se zadani element nalazi u stablu, funkcija treba vratiti pokazivač na njega. Ako zadani element ne postoji u stablu, funkcija treba vratiti NULL. Funkcija treba imati prototip:

```
cvor *fun(cvor *glava, long element);
```

```
cvor *fun(cvor *glava, long element) {

    while (glava && (glava->element != element)) {
        if (element > glava->element) glava = glava->desno;
        else glava = glava->lijevo;
    }

    return glava;
}
```

b) Napisati **REKURZIVNU** funkciju koja će vratiti broj elemenata binarnog stabla čija je vrijednost jednaka njihovoj razini. Razina korijena stabla je 1. Funkcija treba imati prototip:

```
int brNaRazini(cvor *glava, long razina);
```

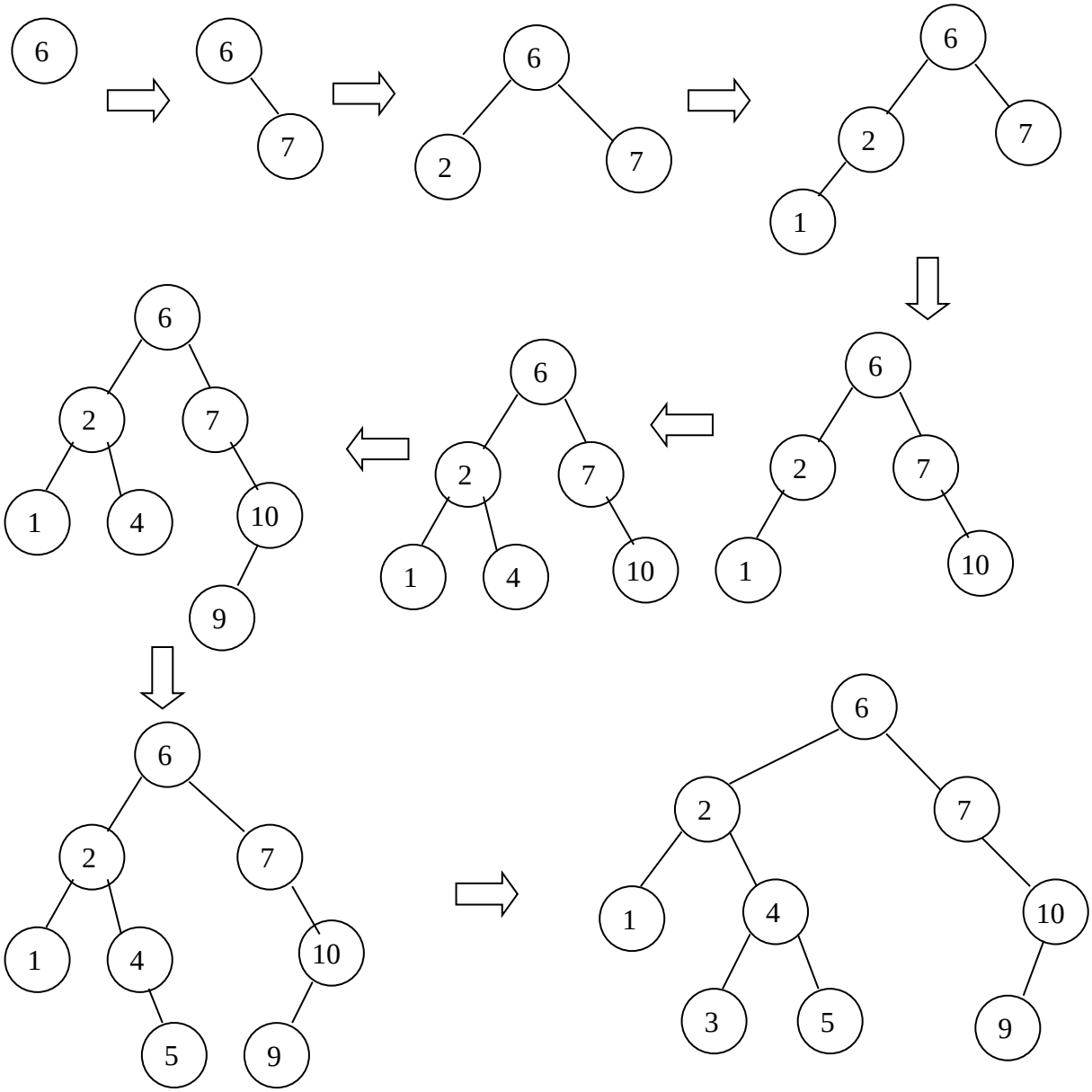
Napisati poziv funkcije iz glavnog programa (samo onu liniju programskog koda kojom se funkcija poziva).

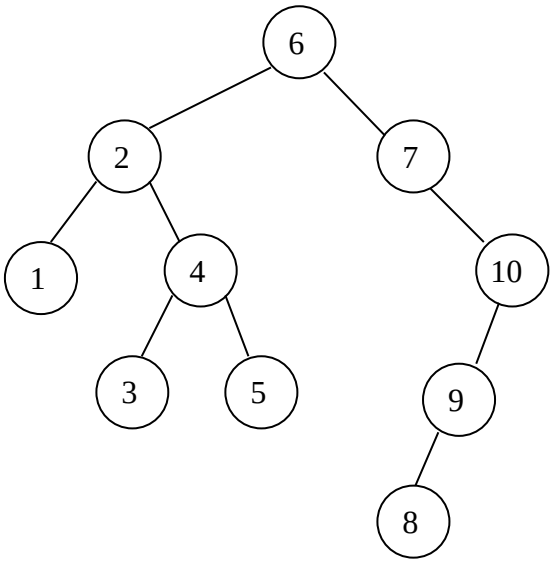
```
int brNaRazini(cvor *glava, int tren_raz){
   if (glava){
      return (glava->broj == tren_raz)
             + brNaRazini (glava->lijevo, tren_raz+1)
             + brNaRazini (glava->desno,  tren_raz+1);
   }
   return 0;
}

brNaRazini(glava, 1);
```

2. Zadan je niz brojeva **6, 7, 2, 1, 10, 4, 9, 5, 3, 8**.

a) Ilustrirati stvaranje sortiranog binarnog stabla (lijevo manji, desno veći) od zadanog niza brojeva. Brojeve u stablo ubacivati s lijeva na desno (počevši od broja 6). Nakon svakog dodavanja novog broja, nacrtati izgled stabla.




6
2
7
1
4
10
3
5
9
8

3. U zadatku treba napisati glavni program koji treba stvoriti listu realiziranu pokazivačima čiji atom sadrži znakovne nizove od 15 elemenata. Nju treba napuniti s proizvoljnih 5 elemenata. Nakon toga napisati funkciju koja preko parametara prima navedenu listu i prepisuje je u stablo te novostvoreno stablo vraća preko imena (return). U glavnom programu stablo ispisati inorder postupkom.

```
Stablo* ListaUStablo(atom* lista)
{
	atom* pomocni;
	Stablo* zaVratiti = new Stablo();
	for(pomocni = lista; pomocni != NULL; pomocni = pomocni->sljed)
	{
		zaVratiti->Dodaj(pomocni->element);
	}
	return zaVratiti;
	//NE raditi delete zaVratiti!!! Memorija mora ostati zauzeta i nakon
izvršavanja funkcije
}

//Rekurzivna verzija
Stablo* ListaUStablo2(atom* lista)
{
	//početni uvijet - jedino mjesto gdje se
	//stvara objekt stabla
	if(lista == NULL)
	{
		Stablo* st = new Stablo();
		return st;
	}
	//prima se pokazivač na stablo iz rekurzivinih poziva
	//koje je u svakom pozivu isti!!! (tj. stablo je jedno jedino
	//pa uvijek ima istu adresu)
	Stablo* st = ListaUStablo2(lista->sljed);
	st->Dodaj(lista->element);
	//vraćanje pokazivača na višu razinu
	return st;
}

int main()
{
	atom* lista=NULL;
	Stablo* st;
	//Funkcija za dodavanje u listu odmah i sortira!!!
	if(!dodaj(&lista, "Pero")) return 0;
	if(!dodaj(&lista, "Ana")) return 0;
	if(!dodaj(&lista, "Marko")) return 0;
	if(!dodaj(&lista, "Ivo")) return 0;
	if(!dodaj(&lista, "Zdravko")) return 0;

	//ispis liste
	ispisi(lista);
	printf("\n\n");
	st = ListaUStablo2(lista);

	//ispis stabla inorder postupkom
	printf("Inorder ispis stabla:\n");
	st->Inorder();

	delete st;
}
```

4. U zadatku treba napisati funkciju koja prebacuje stablo zadano pokazivačima u OOP listu Funkcija preko parametara uzima sve što joj je potrebno i mora biti Rekurzivna. U glavnom programu napuniti stablo proizvoljnim elementima i raditi ispise.

```
void StabloUListu(cvor* stablo, Lista* lista)
{
	if(stablo)
	{
		StabloUListu(stablo->lijevo, lista);
		//ovo punjenje je malo čudno
		//uzima se ascii kod prvog znaka niza (zato jer u listu idu
integeri
		//nije mi se radila nova lista sa stringovima :$
		podatak p;
		p.kljuc = p.element = (int)stablo->element[0];
		lista->DodajNaKraj(p);
		StabloUListu(stablo->desno, lista);
	}
}

int main()
{
	cvor* stablo=NULL;
	Lista* lista = new Lista();

	//Funkcija za dodavanje u sortirano stablo
	stablo=upis(stablo, "Pero");
	stablo=upis(stablo, "Marko");
	stablo=upis(stablo, "Ivo");
	stablo=upis(stablo, "Jakov");
	stablo=upis(stablo, "Stjepan");

	//ispis stabla
	ispisinld(stablo);
	printf("\n\n");

	StabloUListu(stablo, lista);

	//ispis liste
	printf("Ispis liste:\n");
	lista->IspisiListu();

	//sad je pravo vrijeme da se obriše lista iz memorije
	delete lista;

	//Stablo je sortirano, tj. odmah pri upisivanju elemenata isti su
postali sortirani
	//Ako se u rekurziji odabere inorder obilazak stabla (lijevo-desno)
elementi iz sortiranog
	//stabla (lijevo manji, desno veći) će biti uzlazno sortirani u
rezultantnoj listi.
}
```

Ovaj dokument nije službeni. Namjera mu je dati uvid u zadatke koji bi se MOGLI naći na međuispitu. Moguće je da će se u međuispitu javiti i lakši i teži zadaci od navedenih.

JEDNOSTRUKO POVEZANA LISTA

2. (*50 bodova*) U jednostruko povezanu listu spremaju se podaci o studentima: matični broj (**long**), ime i prezime (**50+1 znak**) te prosjek ocjena (**float**). Lista nije sortirana. Napisati funkciju koja će iz liste izbaciti studenta sa zadanim matičnim brojem. Ako je student uspješno izbačen funkcija treba vratiti 1, a ako student nije uspješno izbačen (npr. nije uopće bio u listi) funkcija treba vratiti 0.

3. (*60 bodova*) Na stog realiziran jednostruko povezanom listom spremaju se cjelobrojni podaci (**long**). Napisati funkcije za stavljanje novog elementa na stog i skidanje elementa s vrha stoga. Napisati dodatnu funkciju koja će učitavati cijele brojeve sve dok se ne učita broj 0 te će ih nakon toga ispisati obrnutim redoslijedom od onog kojim su učitavani. Dodatnu funkciju OBAVEZNO treba realizirati koristeći stog, tj. prije napisane funkcije.

3. U jednostruko povezanu listu spremaju se podaci o jelima na meniju u nekom restoranu: šifra jela (**int**), naziv jela (**50+1 znak**), cijena (**float**) i opis jela (**300+1 znak**). Lista je sortirana uzlazno, prema cijeni jela. Napisati funkciju koja će u listu dodati podatke o novom jelu (lista mora ostati sortirana). Funkcija treba imati prototip:

```
int dodaj(zapis **glava, int sif, char naz[], float cijena, char opis[]);
```

4. (70 bodova) U red realiziran jednostruko povezanom listom spremaju se podaci cjelobrojnog tipa (**int**). Napisati funkcije za stavljanje u red i skidanje iz reda. Funkcije trebaju vratiti 1 ako je operacija obavljena uspješno, a 0 inače.

Napisati dodatnu funkciju koja će sve elemente zadanog reda slučajno rasporediti u dva nova reda (koje treba vratiti u glavni program). Svi redovi trebaju biti realizirani jednostruko povezanom listom. Prilikom realizacije dodatne funkcije nije dopušteno izravno pristupati elementima redova, već treba koristiti ranije napisane funkcije za stavljanje u red i skidanje iz reda. Nove, slučajno napunjene, redove treba vratiti u glavni program.

1. (60 bodova) U jednostruko povezanu listu spremaju se podaci o rezultatima izbora: naziv stranke (**30+1 znak**) i broj glasova (**long**). Lista nije sortirana. Napisati funkciju koja će iz liste izbaciti podatke o strankama koje nisu prošle izborni prag te vratiti broj izbačenih zapisa. Funkcija treba imati prototip:

```
int izbori(zapis **glava);
```

Napomena: Stranka nije prošla izborni prag ako ima manje od 5% od ukupnog broja glasova.

2. U jednostruko povezanu listu spremaju se podaci o smještaju studenata u studentske domove: matični broj studenta (**long**), poštanski broj mjesta stanovanja (**int**), šifra

studentskog doma (**short**). Napisati funkciju koja će iz liste izbaciti sve studente koji nisu smješteni u studentski dom (šifra studentskog doma je nula). Funkcija treba vratiti broj izbačenih studenata.

DVOSTRUKO POVEZANA LISTA

2. U red realiziran **dvostruko** povezanom listom spremaju se cjelobrojni podaci (**long**). Napisati funkciju za skidanje jednog podatka iz reda i funkciju za stavljanje novog podatka u red. Napisati glavni program u kojem će se, koristeći prije napisanu funkciju, u red staviti brojevi od 1 do 10.

3. *(60 bodova)* U dvostruko povezanu listu spremaju se cjelobrojni podaci (**long**). Napisati funkciju koja će imati prototip:

```
int izbaciN(zapis **glava, zapis **rep, int N)
```

i koja će iz liste izbaciti prvih **N** elemenata. Ako u listi ima manje od **N** elemenata, funkcija ih treba izbaciti sve. Funkcija treba vratiti stvarni broj izbačenih elemenata.

BINARNA STABLA

1. U sortirano binarno stablo (lijevo manji, desno veći) spremaju se cjelobrojni podaci (**long**).
   - **a)** Nacrtati izgled sortiranog binarnog stabla koje će se dobiti ubacivanjem sljedećih elemenata: 6, 2, 10, 9, 1, 4, 5, 7, 12.
   - **b)** Napisati funkciju koja će zadano stablo ispisati postorder obilaskom.
   - **c)** Odrediti što bi funkcija pod b) ispisala za stablo pod a).

4. U binarno stablo spremaju se podaci o jelima na meniju u nekom restoranu: šifra jela (**int**), naziv jela (**50+1 znak**), cijena (**float**) i opis jela (**300+1 znak**). Stablo je sortirano prema cijeni jela. Napisati funkciju koja će zadano stablo prepisati u jednostruko povezanu listu tako da i lista bude sortirana po cijeni jela (stablo mora ostati nepromijenjeno). Odrediti apriornu složenost funkcije u odnosu na broj elemenata stabla. Funkcija treba imati prototip:

```
void prepisi(cvor *korijen, zapis **glava);
```

Napomena: Prilikom rješavanja ovog zadatka dopušteno je korištenje funkcije iz prethodnog zadatka (čak i ako prethodni zadatak nije riješen).

5. U binarno stablo spremaju se cjelobrojni podaci (**int**). Napisati funkciju koja će pronaći i vratiti koliko se elemenata stabla nalazi na zadanoj razini. Funkcija treba raditi efikasno (ne smije pretraživati čitavo stablo već samo do zadane razine) i treba imati prototip:

```
int funkcija(cvor *korijen, int razina);
```

Napomena: korijen stabla nalazi se na razini 1.

1. Zadan je niz brojeva **6, 7, 2, 1, 10, 4, 9, 5, 3, 8**.

a) *(30 bodova)* Ilustrirati stvaranje sortiranog binarnog stabla (lijevo manji, desno veći) od zadanog niza brojeva. Brojeve u stablo ubacivati s lijeva na desno (počevši od broja 6). Nakon svakog dodavanja novog broja, nacrtati izgled stabla.

4. U binarno stablo spremaju se cjelobrojni podaci (**long**). Stablo je sortirano (lijevo manji, desno veći)

a) *(40 bodova)* Napisati **NEREKURZIVNU** funkciju koja će pronaći zadani element u stablu. Ako se zadani element nalazi u stablu, funkcija treba vratiti pokazivač na njega. Ako se zadani element ne nalazi u stablu, funkcija treba vratiti NULL. Funkcija treba imati prototip:

```
cvor *fun(cvor *glava, long element);
```

b) *(40 bodova)* Napisati **REKURZIVNU** funkciju koja će vratiti broj elemenata binarnog stabla čija je vrijednost jednaka njihovoj razini. Razina korijena stabla je 1. Funkcija treba imati prototip:

```
int brNaRazini(cvor *glava, long razina);
```

Napisati poziv funkcije iz glavnog programa (samo onu liniju programskog koda kojom se funkcija poziva).

1. Zadano je binarno stablo čiji čvorovi sadrže elemente tipa **int**. Napisati funkciju koja će provjeriti jesu li svi elementi u stablu manji ili veći od zadanog broja N. Ako je zadani parametar Op = 0, provjerava se jesu li elementi veći, a ako je zadani parametar Op = 1, provjerava se jesu li elementi manji. Funkcija treba imati prototip:

```
int fun(cvor *korijen, int N, int Op);
```

Funkcija treba vratiti 1 ako je gornji uvjet ispunjen, a 0 inače.

1. U binarnom se stablu pohranjuju cijeli brojevi (**int**). Potrebno je napisati funkciju koja za cijelo stablo provjerava je li cjelobrojna vrijednost svakog roditelja **barem n puta veća** od vrijednosti njegove djece. Funkcija treba imati prototip:

```
int jeliveci(cvor *korijen, int n);
```

Napomena: NIJE dopušteno je korištenje pomoćnih funkcija.

1. U binarno stablo spremaju se cjelobrojni podaci (**int**). Stablo nije sortirano. Napisati funkciju koja će vratiti pokazivač na čvor stabla koji ima najmanju vrijednost. Ako se u stablu nalazi više čvorova koji imaju najmanju vrijednost, funkcija treba vratiti pokazivač na bilo koji od njih. Za prazno stablo funkcija treba vratiti NULL. Funkcija treba imati prototip:

```
cvor *fun(cvor *korijen);
```

1. U binarno stablo spremaju se podaci o rezultatima pismenog ispita: matični broj studenta (**long**), šifra predmeta (**long**), ocjena (**short**) i redni broj izlaska na ispit (**short**). Napisati funkciju koja će ispitati ima li u stablu negativno ocijenjenih ispita

(ocjena 1). Ako se u stablu nalazi barem jedan ispit koji je negativno ocijenjen, funkcija treba vratiti 1, inače funkcija treba vratiti 0. Funkcija treba imati prototip:

```
int fun(cvor *korijen);
```

1. U binarno stablo spremaju se cjelobrojni podaci (**long**). Stablo je sortirano (lijevo manji, desno veći). Napisati funkciju koja će ispisati one elemente stabla koji su manji od aritmetičke sredine svih elemenata stabla. Elemente treba ispisivati od najvećeg prema najmanjem.

Napomena: dopušteno je koristiti pomoćne funkcije.

5. U jednostruko povezano sortirano (lijevo manji, desno veći) stablo spremaju se cjelobrojni podaci (**int**). Napisati NEREKURZIVNU funkciju koja će u stablo dodati novi element. Funkcija treba imati prototip:

```
int dodaj(cvor **korijen, int novi);
```

GOMILE i HEAPSORT

1. Zadan je niz brojeva: **8, 3, 5, 9, 4, 10, 1, 2, 7, 6**. Ilustrirati stvaranje gomile od zadanog niza brojeva (nacrtati izgled gomile nakon svake zamjene dvaju elemenata):

a) algoritmom koji za najgori slučaj ima složenost $O(n \cdot \log_2 n)$

b) algoritmom koji za najgori slučaj ima složenost $O(n)$

2. Zadan je niz brojeva: `4 10 1 7 3 5 4 11 14`
Od navedenih podataka oblikovati gomilu algoritmom koji za najgori slučaj ima složenost O(n) (ne treba ilustrirati korake stvaranja gomile) i nacrtati korake uzlaznog **heapsorta** (nacrtati izgled stabla nakon zamjene svaka dva elementa).

4. Zadan je niz brojeva: **6, 1, 10, 3, 2, 8, 4, 7, 9, 5**.

a) Ilustrirati stvaranje gomile od zadanog niza brojeva. Napisati koja je apriorna složenost korištenog algoritma za najgori slučaj.

b) Ilustrirati postupak heapsort za zadani niz brojeva.

Ispisati izgled polja nakon svake zamjene dvaju elemenata polja.

3. Zadan je niz brojeva: `8 15 10 9 12 7 2 11 14`
Od navedenih podataka oblikovati gomilu (ne treba ilustrirati korake stvaranja gomile) i nacrtati korake uzlaznog **heapsorta** (nacrtati izgled stabla nakon zamjene svaka dva elementa).

4. Gomila koja sadrži cjelobrojne podatke (**int**) zapisuje se u polje. Zadane su funkcije `podesi` (funkcija podešava potpuno binarno stablo u kojem je svojstvo gomile narušeno

samo u korijenu) i `stvoriGomilu` (funkcija stvara gomilu iz zadanog polja). Koristeći zadane funkcije napisati novu funkciju koja će algoritmom heapsort sortirati zadano polje cijelih brojeva. Zadane funkcije imaju prototipove:

```
void podesi(int polje[], int N);
void stvoriGomilu(int polje[], int N);
```