ELEKTROTEHNIČKI FAKULTET
BANJA LUKA

PROGRAMIRANJE II (19.06.2018.)

❶ Neka je definisan tip:

```
typedef struct prop { char key[32];
  char value[128]; } PROPERTY;
```

kojim se reprezentuje jedan konfiguracioni parametar, a koji zapravo predstavlja uređeni par ključ-vrijednost.

Napisati funkciju koja iz ulazne tekstualne datoteke (*f*) čita konfiguracione parametre, te formira i vraća dinamički niz parametara, a čiji je prototip:

```
PROPERTY* read(FILE *f, int *pn);
```

Funkcija preko parametra *pn* vraća ukupan broj pročitanih parametara. U svakom redu ulazne tekstualne datoteke upisan je jedan konfiguracioni parametar u obliku:

kljuc=vrijednost

pri čemu je: *kljuc* – string koji ne sadrži bjeline, a *vrijednost* – string koji može da sadrži bjeline.

Napisati funkciju koja *bubble-sort* algoritmom sortira (rastuće) niz konfiguracionih parametara po ključu, a čiji je prototip:

```
void sort(PROPERTY *array, int n);
```

❷ Napisati funkciju koja pretražuje (binarna pretraga) sortiran niz konfiguracionih parametara (tip PROPERTY definisan u prethodnom zadatku), i na osnovu zadatog ključa vraća traženu vrijednost. Funkcija vraća vrijednost NULL u slučaju da u nizu ne postoji konfiguracioni parametar sa zadatim ključem. Prototip funkcije je:

```
const char* find(PROPERTY *array, int n,
                 const char *key);
```

Vrijednosti konfiguracionih parametara mogu da sadrže specijalne znakove (#), koje treba zamijeniti konkretnim vrijednostima (string). Napisati funkciju sa promjenljivim brojem parametara čiji je prototip:

```
void format(char *str, const char *value, ...);
```

koja formira string *str* na osnovu vrijednosti konfiguracionog parametra *value* pri čemu svaki znak # mijenja sljedećim stringom iz liste varijabilnih parametara. Na primjer:

```
char r[100];
format(r, "Min. vrijednost je #, a max. je #.",
       "100", "200");
printf(r);
// Ispis:
// Min. vrijednost je 100, a max. je 200.
```

❸ Neka je definisan tip:

```
typedef struct node {
  void *data; // dinamički podatak
  struct node *next; } NODE;
```

kojim se reprezentuje čvor jednostruko povezane liste podataka proizvoljnog tipa.

Napisati funkciju koja dodaje novi podatak (*data*) na kraj liste, a čiji je prototip:

```
void add(NODE **phead, void *data);
```

Napisati funkciju koja iz ulazne tekstualne datoteke čita cijele brojeve i, korištenjem definisanog tipa i funkcije *add*, formira (i vraća) listu koja se sastoji od listi cijelih brojeva tako da se u svakoj od tih listi cijelih brojeva nalaze brojevi koji su upisani u jednom redu ulazne datoteke. Pri tome, na početku svakog reda ulazne datoteke, upisan je ukupan broj cijelih brojeva u tom redu. Prototip funkcije je:

```
NODE* read(FILE *f);
```

Pretpostaviti da je ulazna tekstualna datoteka pravilno formatirana. Na primjer, ako je sadržaj ulazne tekstualne datoteke:

```
3 1 -2 3
2 4 5
4 6 -7 8 -9
```

tada funkcija *read* treba da formira i vrati listu koja se sastoji od tri liste cijelih brojeva. Prva lista treba da sadrži brojeve 1, -2 i 3, druga treba da sadrži brojeve 4 i 5, a treća treba da sadrži brojeve 6, -7, 8 i -9.

❹ Neka je definisan tip:

```
typedef struct node {
  void *data; // dinamički podatak
  struct node *left, *right; } NODE;
```

kojim se reprezentuje čvor stabla binarne pretrage (BST) podataka proizvoljnog tipa, te tip:

```
typedef struct bst {
  NODE *root; // korijen BST-a
  int n;      // trenutni broj cvorova
} BST;
```

kojim se reprezentuje BST.

Napisati funkciju koja dodaje novi podatak (*data*) u BST, a čiji je prototip:

```
int add(BST *bst, void *data, int (*cmp)(const
void *, const void *));
```

pri čemu je *cmp* pokazivač na funkciju koja poredi dva podatka nekog tipa, te vraća vrijednost <0 ako je prvi podatak manji od drugog, 0 ako su podaci jednaki, a >0 ako je prvi podatak veći od drugog. Ukoliko se pokuša dodati podatak koji već postoji u stablu, ignorisati pokušaj dodavanja. Funkcija vraća informaciju o tome da li je podatak dodan u stablo: 0 – podatak nije dodan u stablo, 1 – podatak je dodan u stablo.

Napisati funkciju koja obilazi BST u *inorder* redoslijedu, a čiji je prototip:

```
void inorder(BST *bst, void (*f)(const void*));
```

pri čemu je *f* pokazivač na funkciju kojoj se informacioni sadržaj čvora prosljeđuje na obradu (bez modifikacije).

❺ Neka je data sekvencijalna reprezentacija steka i prototipovi funkcija za rad sa stekom:

```
typedef struct stek {int niz[MAX], tos;} STEK;
int push(STEK *s, int podatak);
int pop(STEK *s, int *podatak);
```

kao i sekvencijalna reprezentacija reda i prototipovi funkcija za rad sa redom:

```
typedef struct red {int niz[MAX], f, r;} RED;
int dodaj(RED *red, int podatak);
int obrisi(RED *red, int *podatak);
```

Napisati funkciju koja će na osnovu niza stekova koji je argument funkcije kreirati novi red koristeći pritom pravilo da se kao novi element u redu uvijek dodaje samo parni element sa steka. Obrada stekova se vrši redoslijedom kako su postavljeni u niz. Prototip funkcije je:

```
RED *spoji(STEK *niz, int n);
```

❻ Neka je data matrična reprezentacija grafa pri čemu je sadržaj čvora grafa riječ. Definisati strukturu *GRAF* te napisati funkciju čiji je prototip:

```
void recenica(GRAF *g, int start, char **r);
```

Funkcija treba da kreira rečenicu od riječi (razdvojenih razmakom) koje se nalaze u čvorovima koji se posjećuju prilikom DFS obilaska grafa, pri čemu je *start* indeks početnog čvora u obilasku. Kreiranu rečenicu potrebno je upisati na odgovarajuću memorijsku lokaciju, koristeći parametar *r*. Napisati glavni program u kojem je potrebno kreirati jedan graf sa pet čvorova, te na standardni izlaz, koristeći funkciju *recenica*, ispisati minimalno pet različitih rečenica.

Maksimalan broj bodova po zadacima

Integralno				
1.	2.	5.	6.	Σ
25	25	25	25	**100**

K2				
3.	4.	5.	6.	Σ
25	25	25	25	**100**