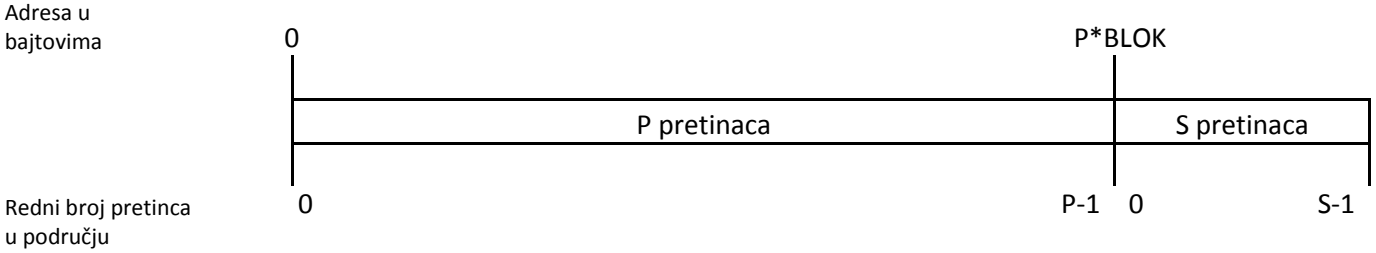
Algoritmi i strukture podataka

19. rujna 2012.

Nije dopušteno korištenje globalnih i statičkih varijabli te naredbe ***goto****.*

1. zadatak (14 bodova)

Tablica raspršenog adresiranja sastoji se od primarnog područja veličine **P** pretinaca i preljevnog područja veličine **S** pretinaca. Oba se područja nalaze u istoj datoteci, prvo primarno područje, a zatim preljevno područje kako je prikazano na slici.



Kapacitet pretinca u oba područja je **C** zapisa. Zapis se sastoji od ključa **k** i cijelog broja **b** a definiran je strukturom

```
typedef struct{
      char k[20];
      int b;
} zapis;
```

Svi potrebni parametri i struktura definirani su u `hash.h` datoteci zaglavlja.

U oba područja zapisi su pohranjeni tehnikom raspršenog adresiranja. Transformacija ključa u adresu obavlja se zadanom funkcijom `int adresa (char *kljuc, char podrucje)`. Funkcija vraća redni broj pretinca u području, iz intervala [0, *brPretinaca* – 1] gdje je *brPretinaca* broj pretin**a**ca u području za koje se određuje adresa. Ulazni parametar `podrucje` može biti `'P'` (primarno područje) ili `'S'` (preljevno područje).

Napisati funkciju `trazi` koja će za zadani ključ vratiti vrijednost cijelog broja **b**. Pretraživanje preljevnog područja mora se obaviti **rekurzivnim** pozivom iste funkcije (funkcije `trazi`). Ako zapis nije pronađen niti u primarnom niti u preljevnom području, funkcija vraća -1. Prototip funkcije je:

```
int trazi(char *kljuc, char podrucje, FILE *f)
```

Napomena: Nerekurzivno rješenje neće se priznavati.

2. zadatak (18 bodova)

a) Napisati funkciju `strcatr`, rekurzivnu inačicu funkcije `strcat`. U svakom rekurzivnom pozivu potrebno je konkatenirati po jedan znak. Možete pretpostaviti da je odredišni niz znakova (destination) dovoljne veličine da se u njega pohrani i znakovni niz koji se konkatenira (source).
Napomena: Nerekurzivno rješenje neće se priznavati.

b) Napisati rekurzivnu funkciju `spoji` koja će s tipkovnice učitavati nizove znakova (maksimalne duljine 20 znakova) sve dok se ne učita niz "`kraj`". Funkcija treba vratiti dva niza znakova:
   1. Niz znakova koji se sastoji od učitanih **brojeva** međusobno odvojenih razmakom
   2. Niz znakova koji se sastoji od učitanih **riječi** međusobno odvojenih razmakom

   Za konkatenaciju nizova znakova potrebno je koristiti funkciju iz **a)** dijela zadatka. Možete pretpostaviti da se riječi sastoje samo od malih slova engleske abecede.
   Nije dozvoljeno unaprijed rezervirati memoriju za nizove koje funkcija mora vratiti.

   Primjer : za ulazne nizove znakova "`prvi`" "`111`" "`222`" "`drugi`" "`333`" "`treci`" funkcija treba vratiti nizove znakova "`111 222 333 `" i "`prvi drugi treci `"

   Napomena: Nerekurzivno rješenje neće se priznavati

c) Napisati glavni program u kojem ćete pozvati funkciju iz **b)** dijela zadatka (funkciju `spoji`)

3. zadatak (10 bodova)

Napisati funkciju koja će od elemenata dviju jednostruko povezanih lista `prva` i `druga` načiniti listu `treca` uzimajući naizmjence elemente iz lista `prva` i `druga`. Listu `treca` funkcija mora vratiti u pozivajući program. 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}

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.

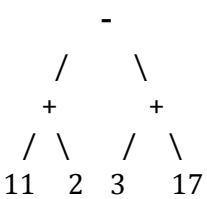
5. zadatak (12 bodova)

U binarnom stablu pohranjena je aritmetička operacija tako da svaki čvor sadrži ili operand ili operator. I operand i operator pohranjeni su kao niz znakova. Operandi su uvijek cijeli brojevi i nalaze se u listovima stabla, a od operatora postoji samo zbrajanje (+) i oduzimanje (-).
Za pretvaranje niza znakova u cijeli broj (int) možete napraviti posebnu funkciju.
Čvor stabla zadan je strukturom:

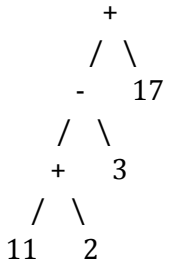
```
typedef struct cvor{
      char *elem;
      struct cvor *lijevo;
      struct cvor *desno;
} cvor;
```

Napisati funkciju koja u pozivajući program vraća rezultat te aritmetičke operacije.

Primjer 1: (11+2)-(3+17) = -7



Primjer 2: (((11+2) -3)+17) = 27



Rješenja:

Zad. 1 (14 bodova)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hash.h"

int trazi(char *kljuc, char podrucje, FILE *f){
	zapis pretinac[C];
	int i=0;
	int adr = adresa(kljuc, podrucje);

	//Ako se radi o preljvnom području adresu tj. rbr. pretinca
	//moramo uvecati za broj pretinaca u primarnom području
	if(podrucje == 'S'){
		adr += P;
	}

	//Pozicioniraj se na trazeni pretinac i procitaj podatke
	fseek(f, adr * BLOK, SEEK_SET);
	fread(pretinac, sizeof(pretinac), 1, f);
	//Prodji kroz sve zapise u pretincu
	for(i=0; i<C; i++){
		if(strcmp(pretinac[i].k, kljuc) == 0){
			//Zapis je nadjen
			return pretinac[i].b;
		}
	}

	//Zapis nije nadjen, trazi u preljevnom podrucju rekurzivno
	//ako smo pretrazivali primarno podrucje. Inace vrati -1 jer
	//zapis nije nadjen
	if(podrucje == 'P'){
		return trazi(kljuc, 'S', f);
	}else{
		return -1;
	}
}
```

Zad 2. (18 bodova)

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

a) (5 bodova)

```c
char *strcatr (char *destination, char *source){
	int duljina = 0;
	if(strlen(source) == 0){
		return destination;
	}else{
		duljina = strlen(destination);
		destination[duljina] = source[0];
		destination[duljina + 1] = '\0';
		return strcatr(destination, source + 1);
	}
}
```

b) (10 bodova)

```c
#define MAX 20

void spoji(char **brojevi, char **rijeci){
	char buff[MAX+1];
	int duljina;
	//Ucitaj niz sa tipkovnice
	scanf("%s", buff);
	if(strcmp(buff, "kraj") == 0){
		return;
	}
	if(buff[0] >= '0' && buff[0] <= '9'){
		//Uneseni niz je broj
		//Realociramo memoriju
		//trenunta duljina niza + duljina unesenog niza + 1 za razmak + 1 za '\0'
		*brojevi = (char*)realloc(*brojevi, strlen(*brojevi) + strlen(buff) + 2);
		strcatr(*brojevi, buff);
		strcatr(*brojevi, " ");
	}else{
		//Uneseni niz je rijec
		//Realociramo memoriju
		//trenunta duljina niza + duljina unesenog niza + 1 za razmak + 1 za '\0'
		*rijeci = (char*)realloc(*rijeci, strlen(*rijeci) + strlen(buff) + 2);
		strcatr(*rijeci, buff);
		strcatr(*rijeci, " ");
	}
	spoji(brojevi, rijeci);
}
```

c) (3 boda)

```c
int main(){
	char *brojevi = NULL;
	char *rijeci = NULL;

	brojevi = (char*)malloc(sizeof(char));
	rijeci = (char*)malloc(sizeof(char));
	brojevi[0] = '\0';
	rijeci[0] = '\0';

	spoji(&brojevi, &rijeci);
	return 0;
}
```

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;
}
```

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);
	}
}
```

Alternativno rješenje – vraća obrnuti poredak poteza

```c
atom *hanoi2(char izvor, char odrediste, char pomocni, int n){
	atom *novi1, *novi2, *pom;
	if(n > 0){
		//Stvaramo novi element
		novi1 = (atom*)malloc(sizeof(atom));
		//Rekurzivni poziv vraca prvi element liste
		novi1->sljed = hanoi2(izvor, pomocni, odrediste, n-1);
		novi1->n = n;
		novi1->t1 = izvor;
		novi1->t2 = odrediste;
		//Rekurzivni poziv - vraca prvi element liste
		novi2 = hanoi2(pomocni, odrediste, izvor, n-1);
		if(novi2 != NULL){
			//Spajamo dvije liste, na kraj liste ciji je prvi element novi2
			//dodajemo listu ciji je prvi element novi (Lista2 -> Lista1)
			pom = novi2;
			while(pom->sljed != NULL){
				pom = pom->sljed;
			}
			pom->sljed = novi1;
			//Vracamo prvi element druge liste
			return novi2;
		}else{
			//Druga lista je prazna pa vracamo prvi element prve liste
			return novi1;
		}
	}else{
		return NULL;
	}
}
```

Zad 5. (12 bodova)

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

typedef struct cvor{
	char *elem;
	struct cvor *lijevo;
	struct cvor *desno;
} cvor;

//Niz znakova koji sadrži broj pretvara u int
int str_to_int(char *str){
	int len = strlen(str);
	int i=0, num=0;
	for(i=len-1; i>=0; i--){
		num += (str[i] - 48) * pow(10., len-i-1);
	}
	return num;
}

int racunaj(cvor *korijen){
	int rez = 0;
	if(korijen->lijevo == NULL && korijen->desno == NULL){
		//List stabla - pretvaramo u int
		return str_to_int(korijen->elem);
	}else{
		if(korijen->elem[0] == '+'){
			//Operacija zbrajanja
			return racunaj(korijen->lijevo) + racunaj(korijen->desno);
		}else if(korijen->elem[0] == '-'){
			//Operacija oduzimanja
			return racunaj(korijen->lijevo) - racunaj(korijen->desno);
		}
	}
}
```