SKUP (klasa set)

http://en.cppreference.com/w/cpp/container/set

Klasa set definisana je u biblioteci set.
SET je kontejner koji implementira veoma korisnu strukturu podataka, eng. red-black tree koja u sebi održava elemente poređane po veličini (ili po nekom našem poretku).

SET je sličan matematičkom pojmu skup. Svaki element je jedinstven. Ubacivanje i uklanjanje elementa sprovodi se za vreme O(log n). Svim elementima može se pristupati (za razliku od priority_queue) i može se iterirati kroz njih pomoću iteratora koji podržavaju ++ i --.

Primer 01

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

inline void ispis(const set<int> &s)
{  set<int>::const_iterator it;
     for(it=s.begin();it!=s.end();it++)
       cout << *it << ' ';
   cout << endl;
}

int main()
{ set <int> s;
  s.insert(3); s.insert(5); s.insert(2);
  s.insert(6); s.insert(1); s.insert(-3);
  ispis(s);
  for(int i=4;i<10;i++) s.insert(i);
  ispis(s);
  s.erase(3);
  s.erase(s.find(7));
  ispis(s);
  s.erase(s.lower_bound(-2));
  ispis(s);
  s.erase(s.upper_bound(5));
  ispis(s);
  cout << "Ima ih: " << s.size() << endl;
  s.clear();
  if (s.empty()) cout << "prazan" << endl;
    return 0;
}
```

IZLAZ

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

```
-3 1 2 4 5 6 8 9
-3 2 4 5 6 8 9
-3 2 4 5 8 9
Ima ih: 6
prazan
```

Primer 02

```
#include <iostream>
#include <set>
#include <iterator>

using namespace std;

int main()
{
	// prazan set uredjen relaciom >
	set <int, greater <int> > skup1;

	// ubacivanje elemenata u skup, proizvoljan poredak ubacivanja
	skup1.insert(40);
	skup1.insert(30);
	skup1.insert(60);
	skup1.insert(20);
	skup1.insert(50);
	skup1.insert(50); // SAMO jedna vrednost 50 ce biti dodata u set
	skup1.insert(10);

	// stampa se sadrzaj skup1
	set <int, greater <int> > :: iterator itr;
	cout << "\nSadrzaj za skup1 je redom : ";
	for (itr = skup1.begin(); itr != skup1.end(); ++itr)
	{
		cout << '\t' << *itr;
	}
	cout << endl;

	// pridruzivanje vrednosti iz skup1 u skup2 sa default poredkom
	set <int> skup2(skup1.begin(), skup1.end());

	// stampa se sadrzaj skup2
	cout << "\nSadrzaj skup2 nakon pridruzivanja clanova iz skup1: ";
	for (itr = skup2.begin(); itr != skup2.end(); ++itr)
	{
		cout << '\t' << *itr;
	}
	cout << endl;

	// ukloni clanove manje od 30 iz skup2
	cout << "\nskup2 nakon uklanjanja clanova manjih od 30 : ";
```

```
	skup2.erase(skup2.begin(), skup2.find(30));
	for (itr = skup2.begin(); itr != skup2.end(); ++itr)
	{
		cout << '\t' << *itr;
	}

	// ukloni clanove cija je vrednost 50 iz skup2
	int num;
	num = skup2.erase (50);
	cout << "\nskup2.erase(50) : ";
	cout << num << " uklonjeni \t" ;
	for (itr = skup2.begin(); itr != skup2.end(); ++itr)
	{
		cout << '\t' << *itr;
	}

	cout << endl;

	//lower bound,  upper bound za set skup1
	cout << "skup1.lower_bound(40) : "
		<< *skup1.lower_bound(40) << endl;
	cout << "skup1.upper_bound(40) : "
		<< *skup1.upper_bound(40) << endl;

	//lower bound, upper bound za set skup2
	cout << "skup2.lower_bound(40) : "
		<< *skup2.lower_bound(40) << endl;
	cout << "skup2.upper_bound(40) : "
		<< *skup2.upper_bound(40) << endl;

	return 0;

}
```

IZLAZ

```
Sadrzaj za skup1 je redom :     60      50      40      30      20      10

Sadrzaj skup2 nakon pridruzivanja clanova iz skup1:     10      20      30      40      50      60

skup2 nakon uklanjanja clanova manjih od 30 :   30      40      50      60
skup2.erase(50) : 1 uklonjeni           30      40      60
skup1.lower_bound(40) : 40
skup1.upper_bound(40) : 30
skup2.lower_bound(40) : 40
skup2.upper_bound(40) : 60
```

MAPA

Klasa map je struktura set<pair<kljuc, vrednost> >

Struktura map koju pretrazujemo pomocu kljuca kojim pristupamo polju vrednost. Za svaki kljuc postoji jedinstvena vrednost. Mapa ima preopterecen operator [] pomocu kog preko kljuca pristupamo polju vrednost. Mapa se nalazi u biblioteci map.

STL Primer01 – mapa

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{   map<string,int> m;
    m["dvadeset"]=20;
    m[string("pet")]=5;

    m.insert(pair<string,int>("dva",2));
    m.insert(make_pair(string("sto"),100));
    m.insert(make_pair("pedeset",50));

map<string,int>::iterator it;
    for(it=m.begin(); it!=m.end();it++)
     cout << it->first << ' ' << it->second << endl;
     return 0;
}
```

IZLAZ
dva 2
dvadeset 20
pedeset 50
pet 5
sto 100

Primer 02

Pekar Joca prodaje burek u **tri oblike** – **četvrtina, polovina**, **tri četvrtine.** Za potrebe proslave skole koja se nalazi u blizini pekare, dobio je porudzbine. Skolska deca su mala niko od njih ne može pojesti ceo burek ali svako je prijavio svojoj uciteljici koliko parce Jocinog bureka može pojesti. Uciteljice instiraju da pekar Joca postuje njihov spisak tako da svako dete dobije **tačnu kolicinu zeljenog bureka.** Napišite program koji će pomoći Joci da odredi koliki je **minimalni** broj bureka koje mora napraviti kako bi svako dete dobilo tačno onoliki komad koliki želi.

Ulaz

U prvom redu standardnog ulaza se nalazi ceo broj N, $1 \leq N \leq 10{,}000$, broj dece koja ce jesti burek.

U svakom od sledećih redova standardnog ulaza nalazi se veličina bureka koju svako od dece želi pojesti tj. razlomak
1/4, **1/2** ili **3/4**.
Izlaz
U prvi i jedini red standardnog ulaza ispisati traženi minimalni broj bureka koje Joca treba da napravi.

PRIMERI
ulaz
3
1/2
3/4
3/4
izlaz
3

ulaz
5
1/2
1/4
3/4
1/4
1/2
izlaz
3

ulaz
6
3/4
1/2
3/4
1/2
1/4
1/2
izlaz
4

Resenje:

```
#include <cstdio>
#include <map>
#include <string>

using namespace std;

map<string, int> broj;

int main( void ) {
  int n;
  scanf( "%d", &n );
  for( int i = 0; i < n; ++i ) {
```

```
    char linija[10];
    scanf("%s", linija );
    broj[linija]++;
  }

  int burek_broj = broj["3/4"] + broj["1/2"] / 2;

  broj["1/4"]  = max( broj["1/4"] - broj["3/4"], 0 );
  broj["1/4"] += 2 * (broj["1/2"] % 2);

  burek_broj += (broj["1/4"] + 3) / 4;

  printf( "%d\n", burek_broj );

  return 0;
}
```

C-mapa

Mapa je apstraktna struktura podataka koja u sebi sadrzi parove oblika (kljuc, vrednost). Pri tom su i kljuc i vrednost unapred odredjenog (ne obavezno istog) tipa (na primer, kljuc nam moze biti string koji predstavlja ime studenta, a vrednost je npr. broj koji predstavlja njegov prosek). Operacije koje mapa mora
efikasno da podrzava su:
_ dodavanje para (kljuc, vrednost) u mapu
_ brisanje para (kljuc, vrednost) iz mape
_ pronalazenje vrednosti za dati kljuc
_ promena vrednosti za dati kljuc

Jedan od najcescih nacina implementacije mape je preko binarnog stabla pretrage.
Svaki cvor ovog stabla sadrzace odgovarajuci par (kljuc, vrednost), tj. svaki cvor ce sadrzati dva podatka. Pri tom ce poredak u stablu biti odredjen poretkom koji je definisan nad kljucevima. Ovim se postize da se pretraga po kljucu moze obaviti na uobicajen nacin, kao i sve ostale navedene operacije.
Jasno je da ovako implementirana mapa nece efikasno podrzavati operaciju pretrage po vrednosti (npr. naci sve kljuceve koji imaju datu vrednost), zato sto poredak
u stablu nije ni na koji na_cin u vezi sa poretkom medju vrednostima. Medjutim, u mapi se najcesce i ne zahteva takva operacija.
Sledi primer koji demonstrira implementaciju mapa pomocu binarnog stabla.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 1024
/* Struktura koja predstavlja cvor stabla */
typedef struct cvor {
char naziv[MAX];
int cena;
struct cvor * levi;
struct cvor * desni;
```

```
} Cvor;
/* Funkcija kreira novi cvor i vraca njegovu adresu */
Cvor * napravi_cvor (char * naziv, int cena)
{
/* dinamicki kreiramo cvor */
Cvor * novi = (Cvor *) malloc (sizeof(Cvor));
/* u slucaju greske ... */
if (novi == NULL)
{
fprintf (stderr,"malloc() greska\n");
exit (1);
}

/* inicijalizacija */
strcpy(novi->naziv, naziv);
novi->cena = cena;
novi->levi = NULL;
novi->desni = NULL;
/* vracamo adresu novog cvora */
return novi;
}
/* Funkcija dodaje novi cvor u stablo sa datim korenom. U cvor se upisuje vrednost (naziv,
cena). Ukoliko naziv
vec postoji u stablu, tada se azurira njegova cena. Cvor se kreira funkcijom napravi_cvor().
Funkcija
vraca koren stabla nakon ubacivanja novog cvora. */
Cvor * dodaj_u_stablo (Cvor * koren, char * naziv, int cena)
{
/* izlaz iz rekurzije: ako je stablo bilo prazno,
novi koren je upravo novi cvor */
if (koren == NULL)
return napravi_cvor (naziv, cena);
/* Ako je stablo neprazno, i koren sadrzi naziv koji je leksikografski manji od datog naziva,
vrednost se umece
u desno podstablo, rekurzivnim pozivom */
if (strcmp(koren->naziv, naziv) < 0)
koren->desni = dodaj_u_stablo (koren->desni, naziv, cena);
/* Ako je stablo neprazno, i koren sadrzi naziv koji je
leksikografski veci od datog naziva, vrednost se umece
u levo podstablo, rekurzivnim pozivom */
else if (strcmp(koren->naziv, naziv) > 0)
koren->levi = dodaj_u_stablo (koren->levi, naziv, cena);
/* Ako je naziv korena jednak nazivu koja se umece, tada se samo
azurira cena. */
else
koren->cena = cena;
/* Vracamo koren stabla */
return koren;
}
```

```
/* Funkcija pretrazuje binarno stablo. Ukoliko pronadje cvor sa vrednoscu naziva koja je jednaka
datom nazivu, vraca adresu tog cvora. U suprotnom vraca NULL */
Cvor * pretrazi_stablo (Cvor * koren, char * naziv)
{
/* Izlaz iz rekurzije: ako je stablo prazno,
tada trazeni broj nije u stablu */
if (koren == NULL) return NULL;
/* Ako je stablo neprazno, tada se pretrazivanje nastavlja u levom ili desnom podstablu, u zavisnosti od toga da li je trazeni naziv respektivno manji ili veci od vrednosti naziva korena. Ukoliko je pak trazeni naziv jednak nazivu korena, tada se vraca adresa korena. */
if (strcmp(koren->naziv, naziv) < 0)
return pretrazi_stablo (koren->desni, naziv);
else if (strcmp(koren->naziv, naziv) > 0)
return pretrazi_stablo (koren->levi, naziv);
else
return koren;
}
/* cvor liste */
typedef struct cvor_liste {
char naziv[MAX];
int cena;
struct cvor_liste * sledeci;
} Cvor_liste;
/* Pomocna funkcija koja kreira cvor liste */
Cvor_liste * napravi_cvor_liste(char * naziv, int cena)
{
Cvor_liste * novi;
if((novi = malloc(sizeof(Cvor_liste))) == NULL)
{
fprintf(stderr, "malloc() greska\n");
exit(1);
}

strcpy(novi->naziv, naziv);
novi->cena = cena;
novi->sledeci = NULL;
return novi;
}

/* Funkcija pronalazi sve nazive cija je cena manja ili jednaka od date, i formira listu koja sadrzi nadjene parove (naziv, cena) u leksikogra-fskom poretku po nazivima. Prilikom pocetnog poziva, treci argument
treba da bude NULL. Funkcija obilazi stablo sa desna u levo, kako bi se prilikom dodavanja na pocetak liste poslednji dodao onaj koji je leksikografski najmanji (tj. on ce biti na pocetku). */
Cvor_liste * pronadji_manje (Cvor * koren, int cena, Cvor_liste * glava)
{
if(koren == NULL)
return glava;
```

```
/* Dodajemo na pocetak liste sve cvorove desnog podstabla cija je cena manja od date. */
glava = pronadji_manje(koren->desni, cena, glava);
/* Dodajemo koren u listu, ako mu je cena manja od date */
if(koren->cena <= cena)
{
Cvor_liste * novi = napravi_cvor_liste(koren->naziv, koren->cena);
novi->sledeci = glava;
glava = novi;
}
/* Dodajemo na pocetak liste sve cvorove levog podstabla cija je cena manja od date */
glava = pronadji_manje(koren->levi, cena, glava);
/* Vracamo glavu liste nakon svih modifikacija */
return glava;
}
/* Funkcija prikazuje listu */
void prikazi_listu(Cvor_liste * glava)
{
if(glava == NULL)  return;
printf("%s = %d\n", glava->naziv, glava->cena);
prikazi_listu(glava->sledeci);
}
/* Funkcija oslobadja listu */
Cvor_liste * oslobodi_listu(Cvor_liste * glava)
{
if(glava == NULL)
return NULL;
glava->sledeci = oslobodi_listu(glava->sledeci);
free(glava);
return NULL;
}

/* Funkcija oslobadja stablo. */
Cvor * oslobodi_stablo (Cvor *koren)
{
if(koren == NULL)
return NULL;
koren->levi = oslobodi_stablo (koren->levi);
koren->desni = oslobodi_stablo (koren->desni);
free(koren);
return NULL;
}
/* test program */
int main(int argc, char ** argv)
{
Cvor * koren = NULL, * pomocni;
Cvor_liste * glava = NULL;
FILE *f;
char naziv[MAX];
int cena;
/* Proveravamo da li je navedeno ime datoteke */
```

```
if(argc < 2)
{
fprintf(stderr, "Morate navesti ime datoteke!\n");
exit(0);
}

/* Otvaramo datoteku */
if((f = fopen(argv[1], "r")) == NULL)
{
fprintf(stderr, "fopen() greska\n");
exit(1);
}
/* Ubacujemo proizvode u stablo */
while(fscanf(f, "%s%d", naziv, &cena) == 2)
{
koren = dodaj_u_stablo(koren, naziv, cena);
}
fclose(f);

/* Testiranje pretrage po nazivu (efikasna operacija) */
printf("Uneti naziv proizvoda koji vas zanima: ");
scanf("%s", naziv);
if((pomocni = pretrazi_stablo(koren, naziv)) == NULL)
printf("Trazeni proizvod ne postoji\n");
else
printf("Cena trazenog proizvoda je: %d\n", pomocni->cena);

/* Testiranje pretrage po ceni (neefikasno) */
printf("Unesite maksimalnu cenu: ");
scanf("%d", &cena);
glava = pronadji_manje(koren, cena, NULL);
prikazi_listu(glava);

/* Oslobadjanje memorije */
glava = oslobodi_listu(glava);
koren = oslobodi_stablo(koren);
return 0;
}
```

Za samostalni rad:
Napisati program koji implementira podsetnik za rođendane. U fajlu se nalazi niz linija oblika:
Ime Prezime DD. MM. YYYY.
tj. za svaku osobu je naveden datum rođenja. Kada se program pokrene, korisnik treba da unese datum, a program treba da pronađe osobu čiji je rođendan najbliži zadatom datumu. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rođendanima uneti u mapu koja je implementirana preko binarnog stabla pretrage.