Univerzitet u Beogradu, Elektrotehnički fakultet

Materijali za vežbe iz
Objektno orijentisanog programiranja

Asistent: Nemanja Kojic, nemanja.kojic@etf.rs

Sadržaj

1. Proširenja programskog jezika C i ne-OO aspekti jezika C++

1.1. Zadatak

Napisati program na programskom jeziku C++ koji ispisuje pozdrav na standardni izlaz.

Rešenje:

```
// pozdrav.C - Ispisivanje pozdrava.
#include <iostream>
using namespace std;
int main () {
    cout << "Pozdrav svima!" << endl;
    return 0;
}
```

Komentar:

Standard za C++ predviđa upotrebu standardnih zaglavlja koji se uključuju bez ekstenzije (**<iostream>**). Mnogi kompajleri podržavaju i prevaziđeni način uključivanja zaglavlja sa ekstenzijom (**<iostream.h>**). Preporučuje se korišćenje prvog načina iz više razloga: (1) nepredvidivo je prevođenje koda ukoliko se koriste zastarela zaglavlja, jer se ne može garantovati da je kompajler implementiran da ih tretira korektno, (2) standardna verzija zaglavlja koristi mehanizam izuzetaka za obradu grešaka, (3) interfejs prema ostatku sistema je jednostavniji i čistiji, (4) poboljšana implementacija standardnih funkcija, (5) podrška za lokalizaciju, (6) podrška za Unicode (enkodovanje pisama koji ne sadrže samo ASCII karaktere), (7) imena su deklarisana u okviru imenskog prostora. Ovo važi generalno za sva standardna zaglavlja i njihove pandane sa ekstenzijom. Sva imena u zaglavlju **iostream** su smeštena u imenskih prostor **std**. Za njihovo uključivanje i direktno korišćenje u programu, koristi se direktiva **using namespace**. Bez ove direktive, potrebno je navoditi punu putanju do deklarisanog imena. Na primer: **std::cout**.

1.2. Zadatak

Napisati na programskom jeziku C++ program koji sa standardnog ulaza učitava niz celih brojeva, izračunava njihov zbir i ispisuje ga na standardni izlaz.

Rešenje:

```
// zbir.C - Zbir niza celih brojeva.
#include <iostream>
using namespace std;
int main () {
    const int DUZ = 100;
    while (true) {
          cout << "\nDuzina niza? ";
          int n; cin >> n;
          if (n <= 0 || n > DUZ) break;
          int a[DUZ]; cout << "Elementi niza? ";
          for (int i=0; i<n; cin >> a[i++]) ;
          int s = 0;
          for (int i=0; i<n; s+=a[i++]) ;
          cout << "Zbir elemenata: " << s << endl;
    }
    return 0;
}
```

Komentar:

Standard za C++ propisuje pravilo da se imena ne moraju deklarisati na početku bloka važenja imena, već se mogu deklarisati bilo gde u programu. Time se deklaracija imena tretira na isti način kao i druge naredbe po gramatici jezika C++. Preporuka je da se imena deklarišu upravo na mestu korišćenja, a ne unapred na početku programa.

1.3. Zadatak

Napisati program na programskom jeziku C++ koji učitava niz celih brojeva sa standardnog ulaza, sortira ga metodom izbora (eng: *selection sort*) i na standardnom izlazu ispisuje sortirani niz. Niz treba da bude smešten u dinamičkoj memoriji i dugačak tačno onoliko koliko je potrebno.

Rešenje:

```
// uredi1.C - Uredjivanje dinamickog niza celih brojeva.
#include <iostream>
using namespace std;
int main () {
    while (true) {
          int n; cout << "Duzina niza? "; cin >> n;
          if (n <= 0) break;
          int* a = new int [n];
          cout << "Pocetni niz? ";
          for (int i=0; i<n; cin >> a[i++]);
          for (int i=0; i<n-1; i++) {
                int& b = a[i]; // b je u stvari a[i].
                for (int j=i+1; j<n; j++)
                      if (a[j]<b) { int c=b; b=a[j]; a[j]=c; }
          }
          cout << "Uredjeni niz: ";
          for (int i=0; i<n; cout << a[i++] << ' ');
          cout << endl;
          delete [] a;
    }
    return 0;
}
```

Komentar:

U jeziku C++ postoje reference. Reference predstavljaju alternativna imena za objekte (i imaju drugačiju semantiku od referenci na jeziku Java). Sve operacije koje se primenjuju na referencu, zapravo se dešavaju nad objektom koji zastupa data referenca. Adresa reference je ista kao adresa objekta. Reference omogućavaju i prenos argumenata u funkcije po reference. Svaka referenca mora biti inicijalizovana na mestu definicije. Jedini izuzetak je ukoliko se ona javlja u deklaraciju formalnog parametra funkcije. Na primer: **`int& b=a[i]`**. Referenca se ne može preusmeriti na drugi objekat, kad se jednom definiše. Ostaje isključivo vezana za objekat kojim je inicijalizovana. Prilikom prenosa objekata (stvarnih parametara) po referenci, na stek poziva se smešta adresa objekta. Sintaksa pristupa objektu preko reference je jednostavnija od sintakse pristupa preko pokazivača. Dakle, referenca na jeziku C++ se može smatrati kao pokazivač sa zgodnijom sintaksom i garantovanom sigurnošću upotrebe.

Za alokaciju i dealokaciju prostora koriste se operatori **`new`** i **`delete`**, umesto bibliotečkih funkcija, kao na jeziku C. Opet, brojne su prednosti korišćenja operatora **`new`** umesto funkcije **`malloc`**. Neke od prednosti su: (1) **`malloc`** nije "type safe" (nema statičke provere tipa od strane kompajlera, već se vrši eksplicitna konverzija void pokazivača od strane programera), (2) malloc vraća **`NULL`** pri neuspešnoj alokaciji, a **`new`** podiže izuzetak **`bad_alloc`**, (3) **`malloc`** je ipak svojstvo jezika C, a new jezika C++ (pri tom, funkcija **`malloc`** ne izaziva poziv konstruktora, već prosto samo alocira prostor bajtova), (4) moguće je redefinsati ponašanje operatora **`new`** i time preuzeti kontrolu nad alokacijom memorije. Ukoliko se za alokaciju memorije koristi operator **`new`**, za dealokaciju se OBAVEZNO mora koristi operator **`delete`**. Nikad se ne smeju mešati operatori i funkcije iz jezika C. Uvek se koristi ili **`new/delete`** ili **`malloc/free`**.

1.4. Zadatak

Napisati funkciju koja prebrojava različite elemente u datom celobrojnom nizu.

Rešenje:

```
// Theme: Basic Structural programming in C/C++
#include <iostream>
int prebrojRazlicite (int* nizZaObradu, int dimenzijeNiza)
//int prebrojRazlicite (int nizZaObradu[], int dimenzijeNiza) - ovo je ekvivalentno
{
    // lokalne promenljive mozemo deklarisati bilo gde unutar bloka
    int razlicitih = 0;
    for (int i=0; i<dimenzijeNiza; i++)
    {
          // ako do kraja ostatka niza nema ponovo istog broja,
          bool nadjen = false;
          for (int j=0; (j<i) && !nadjen; j++)
          if (nizZaObradu[i] == nizZaObradu[j])
          nadjen = true;
          // to znaci da je taj broj razlicit od ostalih elemenata
          if (!nadjen)
          razlicitih++;
    }
    return razlicitih;
}

int main () {
    // Ilustacija korišcenja razlicitih inicijalizatora nizova.
    int a[1] = { 0 }, b[5] = {1,2,1,3,2} , c[5] = {1,1,1,1}; // c[4] je 0
    std::cout<<"Broj razl. u a:"<<prebrojRazlicite(a,sizeof(a)/sizeof(int))<<"\n";
    std::cout<<"Broj razl. u b:"<<prebrojRazlicite(b, sizeof(b)/sizeof(int))<<"\n";
    std::cout<<"Broj razl. u c:"<<prebrojRazlicite(c, sizeof(c)/sizeof(int))<<"\n";
    return 0;
}
```

Komentari:

U deklaraciji funkcije **`prebrojRazlicite`** date su dve opcije sa deklarisanje parametera nizovnog tipa. Moguće je formalni parametar deklarisati kao: (1) pokazivač na prvi element niza (**`int*`**) ili (2) korišćenjem uglastih zagrada (koje implicitno označavaju niz, **`int[]`**). Na jeziku C++ moguće je deklarisati lokalne promenljive bilo gde u izvornom kodu. Promenljive koje se koriste u telu for petlje mogu se definisati u njenom inicijalizacionom delu. Na primer: **`for (int i=0; i<n; i++)`**.

Ukoliko se ne uključe imena iz imenskog prostora direktivom **`using namespace`**, imenima je moguće pristupati navođenjem pune putanje. Na primer: **`std::cout`**.

Operator **`sizeof`** vraća veličinu podatka u bajtovima. Može da se primeni na promenljive ili na identifikatore tipa. Ukoliko se kao operand prosledi identifikator statičkog niza, operator **`sizeof`** vraća dužinu niza u bajtovima. To, međutim, nije moguće uraditi za dinamičke nizove!

1.5. Zadatak

Napisati na programskom jeziku C++ funkciju koja iz datog celobrojnog niza izbacuje elemente koji su jednaki zadatoj vrednosti. Napisati glavni program koji testira rad date funkcije.

Rešenje:

```
// Theme: Basic Structural programming in C/C++
#include <iostream>
using namespace std;
void izbaciJednake (int niz[], int *dimNizaAdr, int vrednostZaIzbaciti)
{
    // preko postojeceg sadrzaja niza prepisujemo samo elemente
    // cija je vrednost razlicita od vrednosti koju treba izbaciti
    int j = 0;
    for (int i=0; i<*dimNizaAdr; i++)
          if (niz[i] != vrednostZaIzbaciti) niz[j++]=niz[i];
    // indeks sledeceg slobodnog mesta u nizu ujedno je dimenzija rezultantnog niza
    *dimNizaAdr=j;
}

int main () {
    int niz1[1] = {0}, niz2[5] = {1,2,1,3,2}, niz3[5] = {1,1,1,1,1};

    int   dimNiz1 = sizeof(niz1)/sizeof(int),
          dimNiz2 = sizeof(niz2)/sizeof(int),
          dimNiz3 = sizeof(niz3)/sizeof(int);
    cout<<"\n\n";
    cout<<"Niz1:\n";
    for (int i=0; i<dimNiz1; i++) cout<<niz1[i]<<" ";
    cout<<"\n";
    izbaciJednake(niz1,&dimNiz1,3);
    cout<<"Niz1 posle brisanja elemenata sa vrednoscu 3:\n";
    for (int i=0; i<dimNiz1; i++) cout<<niz1[i]<<" ";
    cout<<"\n\n";

    cout<<"Niz2:\n";
    for (int i=0; i<dimNiz2; i++) cout<<niz2[i]<<" ";
    cout<<"\n";
    izbaciJednake(niz2,&dimNiz2,2);
    cout<<"Niz2 posle brisanja elemenata sa vrednoscu 2:\n";
    for (int i=0; i<dimNiz2; i++) cout<<niz2[i]<<" ";
    cout<<"\n\n";

    cout<<"Niz3:\n";
    for (int i=0; i<dimNiz3; i++) cout<<niz3[i]<<" ";
    cout<<"\n";
    izbaciJednake(niz3,&dimNiz3,1);
    cout<<"Niz3 posle brisanja elemenata sa vrednoscu 1:\n";
    for (int i=0; i<dimNiz3; i++) cout<<niz3[i]<<" ";
    cout<<"\n";
    return 0;
}
```

1.6. Zadatak

Napisati funkciju koja sabira dva pozitivna cela broja u "neograničenoj tačnosti". Brojevi su predstavljeni kao nizovi decimalnih cifara.

Rešenje:

```
// Theme: Basic Structural programming in C/C++
#include <iostream>
using namespace std;
void saberi(const int* nizCifara1, int duz1,const int* nizCifara2, int duz2,
                int* nizCifaraZbir, int& duzNizCifaraZbir) {
    duzNizCifaraZbir=0;
    // sabiramo cifre u razredima koji postoje u oba broja
    int i = 0, prenos = 0;
    for ( ; (i<duz1) && (i<duz2); i++) {
          int medjuzbir = nizCifara1[i] + nizCifara2[i] + prenos;
          nizCifaraZbir[duzNizCifaraZbir++] = medjuzbir % 10;
          prenos = medjuzbir/10;
    }
    // zatim, ako prvi broj ima vise cifara, cifre iz prvog broja
    for ( ; i<duz1; i++) {
          int medjuzbir = nizCifara1[i]+prenos;
          nizCifaraZbir[duzNizCifaraZbir++]=medjuzbir % 10;
          prenos=medjuzbir/10;
    }
    // zatim, ako drugi broj ima vise cifara, cifre iz drugog broja
    for (; i<duz2; i++) {
          int medjuzbir=nizCifara2[i]+prenos;
          nizCifaraZbir[duzNizCifaraZbir++]=medjuzbir % 10;
          prenos=medjuzbir/10;
    }

    // u svakom slucaju, na kraju treba proveriti prenos iz najstarijeg razreda
    if (prenos>0)
          nizCifaraZbir[duzNizCifaraZbir++]=prenos;
}

int main () {
    const int dimA=1, dimB=5;
    int   a[dimA] = { 0 }, b[dimB] = { 9, 9, 9, 9, 9 },
          c[6]; // maxDimC = max(dimA,dimB)+1
    int dimC;
    cout<<"\n\n";
    saberi(a,dimA,b,dimB,c,dimC);
    for (int i=dimA-1; i>=0; i--) cout<<a[i]; cout<<'+';
    for (int i=dimB-1; i>=0; i--) cout<<b[i]; cout<<'=';
    for (int i=dimC-1; i>=0; i--) cout<<c[i];
    cout<<"\n\n";
    saberi(b,dimB,a,dimA,c,dimC);
    for (int i=dimB-1; i>=0; i--) cout<<b[i]; cout<<'+';
    for (int i=dimA-1; i>=0; i--) cout<<a[i]; cout<<'=';
    for (int i=dimC-1; i>=0; i--) cout<<c[i];
    cout<<"\n\n";
    saberi(b,dimB,b,dimB,c,dimC);
    for (int i=dimB-1; i>=0; i--) cout<<b[i]; cout<<'+';
    for (int i=dimB-1; i>=0; i--) cout<<b[i]; cout<<'=';
    for (int i=dimC-1; i>=0; i--) cout<<c[i];
    cout<<"\n\n";
    return 0;}
```

1.7. Zadatak

Napisati program na programskom jeziku C++ koji iz teksta uklanja suvišne znakove razmaka iz teksta koji se unosi sa standardnog ulaza.

Rešenje:

```
// razmak.cpp - Izostavljanje suvisnih razmaka medju recima.

#include <iostream>
using namespace std;

int main () {
  int znak; bool ima = true;

  while ((znak = cin.get ()) != EOF)     // while (cin.get (znak))
    if (znak != ' ' && znak != '\t')
      { cout.put (znak); ima = znak == '\n'; }
    else if (! ima) { cout.put (' '); ima = true; }
  return 0;
}
```

Komentari:

Objekat cin pored prenosa sa konverzijom omogućava i učitavanje podataka bez konverzije. U ovom slučaju, sa ulaza se čita znak po znak.

U izrazu (`ima = znak == '\n'`) nisu potrebne zagrade, jer operator za dodelu vrednosti ima jedan od najnižih prioriteta.

Objekat **`cout`**, takođe pored prenosa podatak sa konverzijom omogućava u prenos podataka bez konverzije. U ovom zadatku pozivom metode **`cout.put(' ')`** omogućava se slanje jednog znaka na standardni izlaz.

1.8. Zadatak

Napisati program na programskom jeziku C++ koji unosi imena gradova sa standardnog ulaza i uređuje ih po leksikografskom poretku.

Rešenje:

```
// gradovi.cpp - Uređivanje imena gradova smeštenih u dinamičku matricu.

#include <string.h>
#include <iostream>
using namespace std;

const int MAX_GRAD = 100;
const int MAX_DUZ  = 30;

int main () {

  // Citanje i obrada pojedinacnih imena:
  cout << "\nNeuredjeni niz imena gradova:\n\n";
  char** gradovi = new char* [MAX_GRAD];
  int  br_grad=0;
  do {

    char* grad = new char [MAX_DUZ];
    cin.getline (grad, MAX_DUZ);    // Citanje sledeceg imena.
    int duz = strlen (grad);

  // Kraj ako je duzina imena nula.
  if (!duz) { delete [] grad; break; }

    char* pom = new char [duz+1];    // Skracivanje niza.
    strcpy (pom, grad);
    delete [] grad; grad = pom;
    cout << grad << endl;            // Ispisivanje imena.

    // Uvrstavanje novog imena u uredjeni niz starih imena:
    int i;
    for (i=br_grad-1; i>=0; i--)
      if (strcmp (gradovi[i], grad) > 0)
        gradovi[i+1] = gradovi[i];
      else break;
    gradovi[i+1] = grad;

    // Nastavak rada ako ima jos slobodnih vrsta u matrici.
  } while (++br_grad < MAX_GRAD);

  char** pom = new char* [br_grad];  // Skracivanje niza.
  for (int i=0; i<br_grad; i++) pom[i] = gradovi[i];
  delete [] gradovi; gradovi = pom;

  // Ispisivanje uredjenog niza imena:
  cout << "\nUredjeni niz imena gradova:\n\n";
  for (int i=0; i<br_grad; cout<<gradovi[i++]<<endl);
  return 0;
}
```

1.9. Zadatak

Napisati program na programskom jeziku C++ koji određuje polarne koordinate tačaka.

Rešenje:

```
// polar.C - Odredjivanje polarnih koordinata tacke.

#include <math.h>

// Pomocu upucivaca.
void polar (double x, double y, double& r, double& fi) {
  r  = sqrt (x*x + y*y);
  fi = (x==0 && y==0) ? 0 : atan2 (y, x);
}

// Pomocu pokazivaca.
void polar (double x, double y, double* r, double* fi) {
  *r  = sqrt (x*x + y*y);
  *fi = (x==0 && y==0) ? 0 : atan2 (y, x);
}

#include <iostream>
using namespace std;

int main () {
  while (true) {
    double x, y; cout << "\nx,y?  "; cin >> x >> y;
  if (x == 1e38) break;
    double r, fi;
    polar (x, y, r, fi);
    cout << "r,fi: " << r << ' ' << fi << endl;
    polar (x, y, &r, &fi);
    cout << "r,fi: " << r << ' ' << fi << endl;
  }
  return 0;
}
```

Komentari:

Program ilustruje nekoliko aspekata programskog jezika C/C++. Date su dve varijante funkcije koja radi istu stvar. Novina u jeziku C++ je operator za referenciranje (&). Deklarisana referenca sadrži adresu objekta u memoriji, ali se sintaksno ponaša kao sam taj objekat. Nisu potrebna nikakva dereferenciranja prilikom pristupa datom objektu. Ne treba mešati deklaraciju reference od upotrebe istog operatora, samo u drugom kontekstu, kada se dohvata adresa objekta u memoriji.

Primetiti da se u datom rešenju include direktive navode tačno tamo gde su potrebne, a ne sve na početku fajla sa izvornim kodom.

1.10. Zadatak

Napisati program na programskom jeziku C++ koji računa površinu trouglova zadatih dužinama stranica. Program treba da ispisuje interaktivni meni, koji omogućava korisniku da odabere vrstu trougla, a zatim i odgovarajuće stranice, a potom dobije sračunatu površinu.

Rešenje:

```
// trougao1.cpp - Izracunavanje povrsine trougla.

#include <math.h>
#include <iostream>
using namespace std;

// Opsti trougao.
double P (double a, double b, double c) {
  if (a>0 && b>0 && c>0 && a+b>c && b+c>a && c+a>b) {
    double s = (a + b + c) / 2;
    return sqrt (s * (s-a) * (s-b) * (s-c));
  } else return -1;
}

// Jednakokraki trugao.
inline double P (double a, double b) { return P (a, b, b); }

// Jednakostranicni trougao.
inline double P (double a) { return P (a, a, a); }

// Glavna funkcija.
int main () {
  while (true) {
    cout << "Vrsta (1, 2, 3, 0)? "; int k; cin >> k;
  if (k<1 || k>3) break;
    double a, b, c, s;
    switch (k) {
      case 1: cout << "a? ";     cin >> a;
        s = P (a);       break;
      case 2: cout << "a,b? ";   cin >> a >> b;
        s = P (a, b);    break;
      case 3: cout << "a,b,c? "; cin >> a >> b >> c;
        s = P (a, b, c); break;
    }
    if (s > 0) cout << "P= " << s << endl;
      else     cout << "Greska!\n";
  }
  return 0;
}
```

Komentari:

Program ilustruje reupotrebljivost napisanog koda i **`inline`** funkcije. Takođe prikazana je i implementacija konzolnog interaktivnog menija.

1.11. Zadatak

Na programskom jeziku C++ napisati paket funkcija koje omogućavaju rad sa strukturom podataka reda za čekanje i glavni program kojim se testira ispravnost datih funkcija.

Rešenje

Program će biti razložen na 3 fajla:

red.h – zaglavlje koje sadrži prototipove funkcija i definicije elementarnih funkcija

red.cpp – fajl u kojem se implementiraju složenije funkcije navedene u zaglavlju, i

redt.cpp – fajl koji sadrži definiciju glavnog programa u kojem se testira ispravnost napisanih funkcija

```
/*************************************************************
Program:      Redovi celih brojeva
File:         red.h
Description:  Deklaracija paketa za obradu redova celih brojeva.
Author:       Laslo Kraus (lk), Nemanja Kojic (nk)
Environment:  Turbo C++ version 4, 486/66 32mb RAM, DOS 6.0
Notes:        Ovo je pokazni program.
Revisions:    1.00  10/1/2008 (lk) Inicijalna revizija.
              1.01  10/2/2012 (nk) Dodati drugaciji komentari.
*************************************************************/
struct Elem { int broj; Elem* sled; };
struct Red { Elem *prvi, *posl; };

// @brief Stvaranje praznog reda.
inline Red pravi () { Red r; r.prvi = r.posl = 0; return r; }

// @brief Proverava da li je red prazan.
// @param r Red koji treba proveriti
inline bool prazan (const Red& r) { return r.prvi == 0; }

// @brief Dodaje broj na kraj reda.
// @param r Red u koji treba dodati broj
// @param b Broj koji se dodaje na kraj reda.
void dodaj (Red& r, int b);

// @brief Uklanja broj sa pocetka reda.
// @param r Red iz kojeg se uklanja broj
// @returns Broj koji je izbacen iz reda
int uzmi (Red& r);

// @brief Odredjivanje duzine reda.
// @param r Red koji se ispituje
// @returns Duzina niza.
int duz (const Red& r);

// @brief Ispisuje sadržaj reda na standardni izlaz.
// @param r Red koji se ispisuje.
void pisi (const Red& r);

// @brief Brise celokupan sadrzaj reda.
// @param r Red ciji sadrzaj se brise.
void brisi (Red& r);
```

```
/*************************************************************
Program:      Redovi celih brojeva
File:         red.c
Description:  Definicije paketa za obradu redova celih brojeva.
Author:       Laslo Kraus (lk)
       Nemanja Kojic (nk)
Environment:  Turbo C++ version 4, 486/66 32mb RAM, DOS 6.0
Notes:        Ovo je pokazni program.
Revisions:    1.00  10/1/2008 (lk) Inicijalna revizija.
*************************************************************/
#include "red1.h"
#include <stdlib.h>
#include <iostream>
using namespace std;

void dodaj (Red& r, int b) {            // Dodavanje reda na kraju reda.
  Elem* novi = new Elem;
  novi->broj = b; novi->sled = 0;
  if (!r.prvi) r.prvi = novi; else r.posl->sled = novi;
  r.posl = novi;
}

int uzmi (Red& r) {                     // Uzimanje broja s pocetka reda.
  if (!r.prvi) exit (1);
  int b = r.prvi->broj;
  Elem* stari = r.prvi;
  r.prvi = r.prvi->sled;
  if (!r.prvi) r.posl = 0;
  delete stari;
  return b;
}

int duz (const Red& r) {                // Odredjivanje duzine reda.
  int d = 0;
  for (Elem* tek=r.prvi; tek; tek=tek->sled) d++;
  return d;
}

void pisi (const Red& r) {              // Ispisivanje reda.
  for (Elem* tek=r.prvi; tek; tek=tek->sled)
    cout << tek->broj << ' ';
}

void brisi (Red& r) {                   // Praznjenje reda.
  while (r.prvi) {
    Elem* stari = r.prvi; r.prvi=r.prvi->sled; delete stari;
  }
  r.posl = 0;
}
```

```
/*************************************************************
Program:      Redovi celih brojeva
File:         redt.cpp
Description:  Ispitivanje paketa za obradu redova celih brojeva.
Author:       Laslo Kraus (lk)
Environment:  Turbo C++ version 4, 486/66 32mb RAM, DOS 6.0.
Notes:        Ovo je pokazni program.
Revisions:    1.00  10/1/2008 (lk) Inicijalna revizija.
*************************************************************/

#include "red1.h"
#include <iostream>
using namespace std;

int main () {
  Red r = pravi ();
  for (bool dalje=true; dalje; ) {
    cout << "\n1. Dodaj broj        4. Pisi red\n"
            "2. Uzmi broj         5. Brisi red\n"
            "3. Uzmi duzinu       0. Zavrsi\n\n"
            "Vas izbor? ";
    int izb; cin >> izb;
    switch (izb) {
      case 1: int b; cout << "Broj? "; cin >> b; dodaj (r, b); break;
      case 2: if (!prazan(r)) cout << "Broj= " << uzmi (r) << endl;
                else cout << "Red je prazan!\n";
              break;
      case 3: cout << "Duzina= " << duz (r) << endl; break;
      case 4: cout << "Red= "; pisi (r); cout << endl; break;
      case 5: brisi (r); break;
      case 0: dalje = false; break;
      default: cout << "Nedozvoljen izbor!\n"; break;
    }
  }
  return 0;
}
```

2. Klase

2.1. Zadatak

Napisati klasu za redove celih brojeva i glavni program koji pravi objekte klase redova i testira njihovo ponašanje.

Rešenje:

```
// red1.h - Deklaracija paketa za obradu redova celih brojeva.

class Red {
struct Elem { int broj; Elem* sled; };

Elem *prvi;                 // pokazivac na prvi element reda
Elem *posl;                 // pokazivac na poslednji element reda

public:
static Red pravi()          // Stvaranje praznog reda.
{ Red r; r.prvi = r.posl = 0; return r; }
bool prazan () const        // Da li je red prazan?
{ return prvi == 0; }
void dodaj (int b);         // Dodavanje broja na kraju reda.
int uzmi ();                // Uzimanje broja s pocetka reda.
int duz () const;           // Odredjivanje duzine reda.
void pisi () const;         // Ispisivanje reda.
void brisi ();              // Praznjenje reda.
};
```

Komentar:

U klasama se definiše nekoliko nivoa pristupa: **`public, private, protected`**. Ako se ništa ne navede, podrazumeva se **`private`**. Statička metoda **`pravi()`** instancira jedan automatski objekat klase Red koji je prazan (još uvek ne razmatramo konstruktore). Klase definišu strukturu i ponašanje objekata. Strukturu definišu polja, a ponašanje funkcije, koje se zovu metode. Stanje objekta čine vrednosti polja. Metode koje samo čitaju stanje zovu se inspektorske metode, a metode koje menjaju stanje zovu se mutatorske metode. Inspektorske metode se obeležavaju kvalifikatorom **`const`** kojim se garantuje da će kompajler prijaviti svaki pokušaj promene stanja u telu takve metode. Napomena: Da bi ova klasa bila kompletna, budući da ima dinamički alocirane podatke, potrebno je da ima i konstruktor kopije, operator dodele vrednosti i destruktor.

```
// red1.C - Definicije paketa za obradu redova celih brojeva.
#include "red1.h"
#include <stdlib.h>
#include <iostream>
using namespace std;

void Red::dodaj (int b) {               // Dodavanje reda na kraju reda.
Elem* novi = new Elem;
novi->broj = b; novi->sled = 0;
if (!prvi) prvi = novi; else posl->sled = novi;
posl = novi;
}

void Red::pisi () const {               // Ispisivanje reda.
for (Elem* tek=prvi; tek; tek=tek->sled)
   cout << tek->broj << ' ';
}

int Red::uzmi () {                      // Uzimanje broja s pocetka reda.
if (!prvi) exit (1);
int b = prvi->broj;
Elem* stari = prvi;
```

```
prvi = prvi->sled;
if (!prvi) posl = 0; delete stari;
return b;
}

int Red::duz () const {                    // Odredjivanje duzine reda.
int d = 0;
for (Elem* tek=prvi; tek; tek=tek->sled) d++;
return d;
}

void Red::brisi () {                       // Praznjenje reda.
while (prvi) {
   Elem* stari = prvi; prvi=prvi->sled; delete stari;
}
posl = 0;
}


// red1t.C -Ispitivanje paketa za obradu redova celih brojeva sa interaktivnim menijem.
#include "red1.h"
#include <iostream>
using namespace std;

int main () {
Red r = Red::pravi();
for (bool dalje=true; dalje; ) {
    cout << "\n1. Dodaj broj         4. Pisi red\n"
     "2. Uzmi broj          5. Brisi red\n"
     "3. Uzmi duzinu        0. Zavrsi\n\n"
     "Vas izbor? ";
   int izb; cin >> izb;
    switch (izb) {
   case 1: int b; cout << "Broj? "; cin >> b; r.dodaj (b); break;
   case 2: if (!r.prazan()) cout << "Broj= " << r.uzmi () << endl;
        else cout << "Red je prazan!\n";
        break;
   case 3: cout << "Duzina= " << r.duz () << endl; break;
   case 4: cout << "Red= "; r.pisi (); cout << endl; break;
   case 5: r.brisi (); break;
   case 0: dalje = false; break;
   default: cout << "Nedozvoljen izbor!\n"; break;
   }
}
return 0;
}
```

2.2. Zadatak

Na programskom jeziku C++ napisati klasu **Tacka** koja predstavlja tačke u ravni i implementirati metode kojima se definišu osnovne operacije sa tačkama. Napisati i glavni program koji testira rad i ponašanje objekata date klase.

Rešenje:

```
// tacka2.h - Definicija klase tacaka u ravni.
class Tacka {
  double x, y;                        // Koordinate.
public:
  void postavi (double a, double b)   // Postavljanje koordinata.
    { x = a; y = b; }
  double aps () const { return x; }   // Apscisa.
  double ord () const { return y; }   // Ordinata.
  double rastojanje (Tacka) const;    // Rastojanje do tacke.
  friend Tacka citaj ();              // Citanje tacke.
  friend void pisi (Tacka);           // Pisanje tacke.
};
```

```
// tacka2.cpp - Definicije metoda klase tacaka u ravni.
#include "tacka2.h"
#include <cmath>
#include <iostream>
using namespace std;

double Tacka::rastojanje (Tacka t) const   // Rastojanje do tacke.
  { return sqrt (pow(x-t.x,2) + pow(y-t.y,2)); }

Tacka citaj ()                             // Citanje tacke.
  { Tacka t; cin >> t.x >> t.y; return t; }

void pisi (Tacka t)                        // Pisanje tacke.
  { cout << '(' << t.x << ',' << t.y << ')'; }
```

```
// tacka2t.cpp - Ispitivanje klase tacaka u ravni.
#include "tacka2.h"
#include <iostream>
using namespace std;

int main () {
  cout << "t1? "; double x, y; cin >> x >> y;
  Tacka t1; t1.postavi (x, y);
  cout << "t2? "; Tacka t2 = citaj();
  cout << "t1=(" << t1.aps () << ',' << t1.ord ()
       << "), t2="; pisi(t2); cout << endl;
  cout << "Rastojanje=" << t1.rastojanje (t2) <<endl;
  return 0;
}
```

Komentari:

Funkcija **citaj()** je prijateljska da bi direktno mogla da pristupi privatnim članovima klase. Ona stvara automatski objekat. Isto važi za metodu **pisi()**. Metoda **postavi()** koristi se za definisanje stanja objekta (još uvek ne razmatramo konstruktore). Moglo bi i bez prijateljskih funkcija, ali bi bile potrebne inspektorske metode za čitanje koordinata. Metoda **rastojanje(Tacka)** i funkcija **pisi(Tacka)** dobijaju parametar po vrednosti (to podrazumeva kopiranje objekta – jedan od mehanizama jezika C++).

2.3. Zadatak

Napisati na programskom jeziku C++ klasu za rad sa trouglovima i glavni program koji prikazuje njeno funkcionisanje (učitava i sortira trouglove prema površini).

Rešenje:

```
// trougao2.h - Definicija klase trouglova.

#include <cstdlib>
#include <iostream>
using namespace std;

class Trougao {
  double a, b, c;                                   // Stranice trougla.
public:
  static bool moze (double a, double b, double c) { // Provera stranica.
    return a>0 && b>0 && c>0 && a+b>c && b+c>a && c+a>b;
  }

  void postavi (double aa, double bb, double cc) { // Postavljanje
    if (! moze (aa, bb, cc)) exit (1);             //   koordinata.
    a = aa; b = bb; c = cc;
  }
  double uzmiA () const {
  // this->b = 5;  // Ne moze bez eksplicitne konverzije.
  // ((Trougao*)this)->b = 5;    // Sada moze
  return a;
  }               // Dohvatanje stranica.
  double uzmiB () const { return b; }
  double uzmiC () const { return c; }
  double O () const { return a + b + c; }           // Obim trougla.
  double P () const;                                // Povrsina trougla.
  bool citaj ();                                    // Citanje trougla.
  void pisi () const;                               // Pisanje trougla.

  ~Trougao() { cout << "Pozvan destruktor za "; this->pisi(); cout << endl; }
};

// trougao2.C - Definicije metoda klase trouglova.
#include "trougao2.h"
#include <iostream>
#include <cmath>
using namespace std;

double Trougao::P () const {                        // Povrsina trougla.
  double s = O () / 2;
  return sqrt (s * (s-a) * (s-b) * (s-c));
}

bool Trougao::citaj () {                            // Citanje trougla.
  double aa, bb, cc;
  cin >> aa >> bb >> cc;
  if (! moze (aa, bb, cc)) return false;
  a = aa; b = bb; c = cc;
  return true;
}

void Trougao::pisi () const {                       // Pisanje trougla.
  cout << "Troug(" << a << ',' << b << ',' << c << ')';
}
```

```
// trougao2t.cpp - Ispitivanje klase trouglova.

#include "trougao2.h"
#include <iostream>
using namespace std;

int main () {
  cout << "Broj trouglova? "; int n; cin >> n;
  Trougao* niz = new Trougao [n];
  for (int i=0; i<n; ) {
    cout << i+1 << ". trougao? ";
    double a, b, c; cin >> a >> b >> c;
    if (Trougao::moze(a,b,c)) niz[i++].postavi (a, b, c);
      else cout << "*** Neprihvatljive stranice!\n";
  }
  for (int i=0; i<n-1; i++)
    for (int j=i+1; j<n; j++)
      if (niz[j].P() < niz[i].P())
        { Trougao pom = niz[i]; niz[i] = niz[j]; niz[j] = pom; }
  cout << "\nUredjeni niz trouglova:\n";
  for (int i=0; i<n; i++) {
    cout << i+1 << ": "; niz[i].pisi ();
    cout << " P=" << niz[i].P() << endl;
  }
  delete [] niz;

  return 0;
}
```

Komentari:

Svaka metoda klase ima jedan skriveni argument **`this`**. This sadrži adresu objekta za koji je pozvana metoda i omogućava pristup članovima klase unutar metode. Niz sadrži na početku alocirane, ali neinicijalizovane trouglove (poziva se podrazumevani konstruktor). Prilikom sortiranja, algoritam je neefikasan jer vrši fizičko kopiranje objekata prilikom zamene mesta (veoma je osetljiv na veličinu objekata). U tom slučaju imamo još dva mehanizma (dodela vrednosti – operator dodele vrednosti, inicijalizacija drugim objektom – konstruktor kopije).

2.4. Zadatak

Napisati na programskom jeziku C++ klasu za rad sa uglovima, kao i glavni program koji ilustruje korišćenje date klase.

Rešenje:

```
// ugao.h - Definicija klase uglova.
#include <iostream>
using namespace std;

const double FAKTOR = 3.14159265358979323 / 180;

class Ugao {
  double ugao;                          // Ugao u radijanima.
public:                                 // Konstruktori:
  Ugao (double u=0) {                   //  - podrazumevani i konverzije
 ugao = u;
  }

  explicit Ugao (int stp, int min=0, int sek=0) {   //  - na osnovu stepeni.
  ugao = ((sek/60.+min)/60+stp) * FAKTOR;
  cout << "Pozvan je konstruktor konverzije." << endl;
  }

  double rad () const { return ugao; }            // Radijani.

  int stp () const                                // Stepeni.
    { return (ugao / FAKTOR); }

  int min () const                                // Minuti.
    { return int (ugao / FAKTOR * 60) % 60; }

  int sek () const                                // Sekunde.
    { return int (ugao / FAKTOR * 3600) % 60; }

  void razlozi (int& st, int& mi, int& se) const  // Sva tri dela
    { st = stp (); mi = min (); se = sek () ; }   //   odjednom

  Ugao& dodaj (Ugao u)                  // Dodavanje ugla.
    {  ugao += u.ugao; return *this; }

  Ugao& pomnozi (double a)              // Mnozenje realnim brojem.
    { ugao *= a; return *this; }

  void citaj () { cin >> ugao; }        // Citanje u radijanima.

  void citajStepene () {                // Citanje u stepenima.
    int stp, min, sek; cin >> stp >> min >> sek;
    *this = Ugao (stp, min, sek);    // DODELA VREDNOSTI!!
  }

  void pisi () const { cout << ugao; }  // Pisanje u radijanima;

  void pisiStepene () const            // Pisanje u stepenima.
    { cout << '(' << stp() << ':' << min() << ':' << sek() << ')'; }
};
```

```
// ugaot.C - Ispitivanje klase uglova.

#include "ugao.h"
#include <iostream>
using namespace std;

void m1(Ugao u1) {
cout << "Obradjuje se ugao funkcijom m1 "; u1.pisi(); cout << endl;
}

int main () {
  Ugao u1, u2;
  cout << "Prvi  ugao [rad]? "; u1.citaj ();
  cout << "Drugi ugao [rad]? "; u2.citaj ();
  Ugao sr = Ugao (u1).dodaj (u2).pomnozi (0.5); //Podraz. ponasanje konstrukt. kopije
  cout << "Srednja vrednost= "; sr.pisi();      cout << ' ';
                               sr.pisiStep (); cout << endl;

  Ugao u3 = 2;   // Poziva se konstruktor konverzije klase Ugao.
  m1(u3);    // Kopiranje vrednosti objekta u3 u objekat u1 (formalni parametar)
  m1(2);     // Poziva se konstruktor konverzije prilikom prenosa stvarnih parametara.
  return 0;
}
```

Komentari:

Primer prikazuje sve vrste konstruktora klasa (podrazumevani, konverzije, kopije). Ilustruju se mehanizmi pri kojima dolazi do poziva konstruktora. Razlikovati mehanizam inicijalizacije drugim objektom od mehanizma dodele vrednosti jednog objekta drugom. Funkcija **`dodaj(Ugao)`** prihvata argument po vrednosti. Na taj način se omogućava da metoda garantuje da neće promeniti argument ili bi prenos mogao da se izvrši po referenci sa kvalifikatorom **`const`** (**`const Ugao&`**).

2.5. Zadatak

Napisati na programskom jeziku C++ klasu celobrojnih redova ograničenog kapaciteta i glavni program koji prikazuje korišćenje date klase.

Rešenje:

```
// red2.h - Definicija klase redova ogranicenih kapaciteta (kružni bafer).

class Red {
  int *niz, kap, duz, prvi, posl;
public:
  explicit Red (int k=10);      // Stvaranje praznog reda (nije konverzija).
  Red (const Red& rd);          // Stvaranje reda od kopije drugog.
  ~Red () { delete [] niz; }                // Unistavanje reda.
  void stavi (int b);                       // Stavljanje broja u red.
  int uzmi ();                              // Uzimanje broja iz reda.
  bool prazan () const { return duz == 0; } // Da li je red prazan?
  bool pun () const { return duz == kap; }  // Da li je red pun?
  void pisi () const ;                     // Pisanje sadrzaja reda.
  void prazni () { duz = prvi = posl = 0; } // Praznjenje reda.
};


// red2.C - Definicije metoda klase redova ogranicenih kapaciteta.
#include "red2.h"
#include <iostream>
#include <cstdlib>
using namespace std;

Red::Red (int k) {                      // Stvaranje praznog reda.
  niz = new int [kap = k];
  duz = prvi = posl = 0;
}

Red::Red (const Red& rd) {              // Stvaranje reda od kopije drugog.
  niz = new int [kap = rd.kap];
  for (int i=0; i<kap; i++) niz[i] = rd.niz[i];
  duz = rd.duz; prvi = rd.prvi; posl = rd.posl;
}

void Red::stavi (int b) {               // Stavljanje broja u red.
  if (duz++ == kap) exit (1);
  niz[posl++] = b;
  if (posl == kap) posl = 0;
}

int Red::uzmi () {                      // Uzimanje broja iz reda.
  if (duz-- == 0) exit (2);
  int b = niz[prvi++];
  if (prvi == kap) prvi = 0;
  return b;
}

void Red::pisi () const {              // Pisanje sadrzaja reda.
  for (int i=0; i<duz; cout<<niz[(prvi+i++)%kap]<<' ');
}
```

```
// red2t.C - Ispitivanje klase redova ogranicenih kapacieta.
#include "red2.h"
#include <iostream>
using namespace std;

int main () {
  Red* rd = new Red (5); bool kraj = false;
  // Red r1 = 3;   // Ne moze, jer je konstruktor konverzije izmenjen da bude explicit.
  while (! kraj) {
    cout << "\n1. Stavaranje reda\n"
            "2. Stavljanje podatka u red\n"
            "3. Uzimanje podatka iz reda\n"
            "4. Ispisivanje sadrzaja reda\n"
            "5. Praznjenje reda\n"
            "0. Zavrsetak rada\n\n"
            "Vas izbor? ";
    int izbor; cin >> izbor;
    switch (izbor) {
    case 1: // Stvaranje novog reda:
      cout << "Kapacitet? "; int k; cin >> k;
      if (k > 0) { delete rd; rd = new Red (k); }
        else cout << "*** Nedozvoljeni kapacitet! ***\a\n";
      break;
    case 2: // Stavljanje podatka u red:
      if (! rd->pun ()) {
        cout << "Broj?      "; int b; cin >> b; rd->stavi (b);
      } else cout << "*** Red je pun! ***\a\n";
      break;
    case 3: // Uzimanje podatka iz reda:
      if (! rd->prazan ())
        cout << "Broj=      " << rd->uzmi () << endl;
      else cout << "*** Red je prazan! ***\a\n";
      break;

    case 4: // Ispisivanje sadrzaja reda:
      cout << "Red=       "; rd->pisi (); cout << endl;
      break;
    case 5: // Praznjenje reda:
      rd->prazni (); break;
    case 0: // Zavrsetak rada:
      kraj = true; break;
    default: // Pogresan izbor:
      cout << "*** Nedozvoljen izbor! ***\a\n"; break;
    }
  }
  return 0;
}
```

2.6. Zadatak

Napisati na programskom jeziku C++ klasu za apstraktni tip podataka Text koja omogućava rad sa znakovnim nizovima.

Rešenje:

```
/*
 * Tekst.h
 *  Created on: 27.10.2015.
 *      Author: Nemanja Kojic
 */
#ifndef TEXT_H_
#define TEXT_H_

#include <iostream>
using namespace std;

/**
 * This class is ADT for texts.
 */
class Text {
char* stringPtr;

public:
// Default constructor.
Text(): stringPtr(0) { }
// Conversion constructor.
Text(const char*) ;
// Std copy constructor.
Text(const Text&) ;
// Move copy constructor.
Text(Text&&);
// Destructor.
~Text();

// Overloaded operator<< (you must use only this signature).
friend ostream& operator<<(ostream& out, const Text&);

private:
// Standard copy assignment operator - disabled.
Text& operator=(const Text& other) = delete;
// Move assignment operator - disabled.
Text& operator=(Text&& other) = delete;
};

#endif /* TEXT_H_ */
```

```cpp
/*
 * text.cpp
 *  Created on: 27.10.2015.
 *      Author: Nemanja Kojic
 */
#include "text.h"
#include <cstring>
#include <iostream>

using namespace std;

Text::Text (const char* t) {
stringPtr = new char [strlen(t)+1];
strcpy (stringPtr, t);
}

Text::Text (const Text& t) {
stringPtr = new char [strlen(t.stringPtr)+1];
strcpy (stringPtr, t.stringPtr);
// For debugging purpose only - don't use it in real life.
cout << "cc" << endl;
}

Text::Text(Text&& t): stringPtr (t.stringPtr) {
// Clear the state of the object 't'.
t.stringPtr = nullptr;
// For debugging purpose only - don't use it in real life.
cout << "mvcc" << endl;
}

Text::~Text () {
// For debugging purpose only - don't use it in real life.
cout << "D[Text -> \"" << ((stringPtr!=NULL) ? stringPtr : "NULL") << "\"]"<<endl;
delete [] stringPtr;
stringPtr = nullptr;
}

ostream& operator<<(ostream& out, const Text& text) {
return out << text.stringPtr;
}
```

```
//============================================================================
// Name        : main.cpp
// Author      : Nemanja Kojic
// Version     : v1.0
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================
#include "text.h"
#include <iostream>

using namespace std;

Text tmpCreate1(const char* txt) {
return Text(txt);
}

Text tmpCreate2(Text txt) {
return txt;
}

int main() {
Text hello("Hello");   // Poziva se Text(char*).
Text a = hello;        // Poziva se Text(const Text&).
Text b;                // Poziva se Text().
cout << "greeting = " << hello << endl;
cout << "a        = " << a << endl;
// Annonymous temp object passed by ref, and destroyed by the end of the statement.
cout << "temp1    = " << Text("temp text - by ref") << endl;
// Copy elision - returned object used directly as the parameter of operator<< func.
cout << "temp2    = " << tmpCreate1("temp text - by ref, copy elision") << endl;
// Function tmpCreate2 returns auto object by value - result returned by move cc
// and passed to function operator<<.
cout << "temp3    = " << tmpCreate2("temp text - created by calling move cc") << endl;
// Invokes copy constructor - the object destroyed by the end of sentence.
cout << "temp4    = " << Text(a) << endl;
return 0;
}
```

Komentari:

Destruktori omogućavaju oslobađanje resursa u trenutku kada se objekat koji ih zauzima uništava. Objekti na C++ se uništavaju deterministički. Ako su u dinamičkoj memoriji, uništavaju se dejstvom operatora delete. Sa druge strane, ukoliko se radi o lokalnim, automatskim objektima (na proceduralnom steku), njihov životni vek traje sve dok se ne napusti opseg važenja imena u kom su i nastali. Ukoliko se ne definiše eksplicitno destruktor u klasi, kompajler generiše prazan destruktor. Dakle, mehanizam poziva destruktora uvek postoji I dešava se. Eksplicitnim definisanjem destruktora, programer može da udene željeno ponašanje prilikom uništavanje objekta. Najčešće pravilo je da, ako se unutar klase memorija alocira dinamički, takva klasa obavezno mora da ima definisan destruktor u kojem se oslobađa memorija. Slično važi i za ostale resurse (npr. otvorene fajlove). Obratiti pažnju na konstruktor kopije i njegov potpis: `Tekst (const Tekst& t);` Šta bi se desilo, ako bi potpis konstruktora kopije bio ovakav: `Tekst (const Tekst t);` - beskonačna petlja! U glavnom programu, redosled pozivanja destruktora automatskih objekata je obrnut od redosleda njihovog nastanka.

2.7. Konstrukcija i destrukcija objekata i podobjekata

Sledeći primer prikazuje C++ mehanizam pravljenja i uništavanja objekata koji sadrže objekte drugih klasa kao svoje atribute (podobjekte).

Rešenje:

```
/*
 * MyClasses.h
 *  Created on: 03.11.2015.
 *      Author: Nemanja Kojic
*/

#ifndef SRC_MYCLASSES_H_
#define SRC_MYCLASSES_H_

#include <iostream>
using namespace std;

class DataStruct {
public:
DataStruct() { cout << "DS: default ctor" << endl; }

DataStruct(const DataStruct& other) { cout << "DS: copy ctor" << endl; }

DataStruct(DataStruct&& other) { cout << "DS: mv-copy ctor" << endl; }

~DataStruct() { cout << "DS: dtor" << endl; }

DataStruct& operator=(const DataStruct& other)
{ cout << "DS: op=" << endl; return *this; }
};


class MyInt {
DataStruct myStruct; // Nested object!

public:
MyInt() { cout << "MI: default ctor" << endl; }

MyInt(const MyInt& obj) :myStruct(obj.myStruct)
{ cout << "MI: copy ctor" << endl; }

MyInt(MyInt&& other): myStruct(std::move(other.myStruct))
{ cout << "MI: mv-copy ctor" << endl; }

~MyInt() { cout << "MI: dtor" << endl; }
};

#endif /* SRC_MYCLASSES_H_ */
```

```cpp
/*
 * test.cpp
 *  Created on: 03.11.2015.
 *      Author: Nemanja Kojic
 */
#include "myclasses.h"
#include <utility>

/* This function receives its arguments by value. */
void f(MyInt i1) { cout << "Invoked f() " << endl; }

int main() {
// Object i1 created and initialized using the default constructor.
// Its subobject myStruct is initialized using its own default constructor.
// The default constructor of the subobject is invoked before
// MyInt's default constructor's body is executed.
cout << "---- Statement #1 (MyInt i1;): -------------" << endl;
MyInt i1;

// The object is passed as the input parameter to f and the local object
// is initialized using the copy constructor.
// Firstly, the subobject is initilized using its own copy constructor,
// the can be invoked only in the section of initializers".
cout << "---- Statement #2 (f(i1);): ----------------" << endl;
f(i1);

// Force copying the argument using the move copy constructor.
// USE THIS VERY CAREFULLY!
// Object i1 gets cleared!
cout << "---- Statement #3 (f(std::move(i1));): -----" << endl;
f(std::move(i1));

// Destruction of the local objects takes place here.
cout << " ---- End of main (~i1): -------------------" << endl;
return 0;
 }
```

```
---- Statement #1 (MyInt i1;): -------------
DS: default ctor
MI: default ctor
---- Statement #2 (f(i1);): ----------------
DS: copy ctor
MI: copy ctor
Invoked f()
MI: dtor
DS: dtor
---- Statement #3 (f(std::move(i1));): -----
DS: mv-copy ctor
MI: mv-copy ctor
Invoked f()
MI: dtor
DS: dtor
 ---- End of main (~i1): -------------------
MI: dtor
DS: dtor
```

2.8. Zadatak

Trgovačka firma (*Company*) prima narudžbine od proizvoljno mnogo klijenata (*Customer*). Svaki klijent šalje narudžbinu samo jednoj firmi, a narudžbina se sastoji od proizvoljno mnogo zahteva (*Demand*). Jedan zahtev sadrži ime tražene vrste artikla. Klijent postavlja zahtev sa imenom artikla kao parametrom.

Firma ima svoje skladište (*Warehouse*) koje sadrži proizvoljno mnogo vrsta artikala (*Item*). Kada klijent pošalje narudžbinu, ispituje se svaki zahtev ponaosob i u skladištu se traže artikli datog tipa. Ukoliko u skladištu ne postoje artikli datog tipa, kompanija šalje zahtev svom dobavljaču robe (*Supplier*). Dobavljač šalje pošiljku kompaniji, a kompanija prosleđuje traženi artikal klijentu.

Napisati program na jeziku C++ na osnovu zadatog opisa problema. Primeniti konceptualnu dekompoziciju.

Rešenje:

```
// Trgovina.h
#ifndef __KONCEPTI__H__
#define __KONCEPTI__H__

#include <vector>
#include <string>
#include <list>
using namespace std;

class Item
{
public:
Item (const string& n): name(n) {}
const string& getName() const { return name; }
private:
string name;
};

class Customer;

class Demand
{
public:
Demand(Customer* c, const string& itName, unsigned amount=1);
const string& getItemName() const { return itemName; }
void setAmount(unsigned amount) { amountOfItems = amount; }
unsigned getAmount() const { return amountOfItems; }

private:
string itemName;
unsigned amountOfItems;
};

class DeliveryPackage
{
public:
DeliveryPackage (Demand *d) : myDemand(d)
{
   items = new list<Item*>();
}
void putItem   (Item *item) { items->push_back(item); }
list<Item*>& getItems() const { return *items; }
~DeliveryPackage() { delete items; myDemand = NULL; }

private:
Demand *myDemand;
```

```
list<Item*> *items;
};

class Supplier
{
public:
DeliveryPackage* receiveDemand(Demand* d);
};

class Warehouse;

class Company
{
public:
Company (const string& cName);
void receiveDemand(Customer *c, Demand *d);
void setSupplier(Supplier* s) { mySupplier = s; }
const string& getName() const { return name; }
~Company();

friend void populateItems (Company* c);
private:
Company (const Company& c) {} // Constructor copy disabled.
Company& operator= (const Company& c) {} // Assignment disabled.
string name;
Warehouse *myWarehouse;
Supplier *mySupplier;
};

class Warehouse
{
public:
void addItem(Item* it) { myItems.push_back(it); }
void removeItem(Item *it) { /* TODO: Implement this method! */ }

const vector<Item*>& getItems() const { return myItems; }
Item* checkItem(const string& itemToCheck);
private:
vector<Item*> myItems;
};

class Customer
{
public:
Customer(const string& cName) : name(cName) {}
void addDemand(Demand* d) { myDemands.push_back(d); }
void createDemand(const string& itName);
void sendOrder();
void notify(Item* itemFound);
void setCompany(Company* c) { myCompany = c; }
~Customer();
private:
string name;
vector<Demand*> myDemands;
Company* myCompany;
};

#endif

// Trgovina.cpp
```

```
#include "trgovina.h"
#include <cstdlib>
#include <iostream>
using namespace std;

void Company::receiveDemand(Customer* c, Demand *d)
{
Item* itemFound = myWarehouse->checkItem(d->getItemName());
if (itemFound)
   c->notify(itemFound);
else
{
   const int SUPPLY_QUOTA = 5;
   cout << "Item \""<< d->getItemName() << "\" is out of stock!" << endl;

   Demand *supplyDemand = new Demand(*d);
   supplyDemand->setAmount(d->getAmount() * SUPPLY_QUOTA);

   DeliveryPackage *dp = mySupplier->receiveDemand(supplyDemand);
   list<Item*> items = dp->getItems();
   cout << "Company \"" << getName() << "\" received "
        << supplyDemand->getAmount() << " \"" <<d->getItemName()
        << "\" items." << endl;

   if (!items.empty())
   {
      c->notify(items.front());
      items.pop_front();
   }

   while (!items.empty())
   {
      myWarehouse->addItem(items.front());
      items.pop_front();
   }

   delete supplyDemand;
   delete dp;
}
}

Demand::Demand(Customer* c, const string& itName, unsigned amount)
: itemName(itName), amountOfItems(amount)
{
c->addDemand(this);
}

Company::Company(const string& cName): name(cName)
{
myWarehouse = new Warehouse();
mySupplier = new Supplier();
}
```

```
Company::~Company()
{
delete myWarehouse;
delete mySupplier;
}

void Customer::createDemand(const string& itemName)
{
Demand* d = new Demand(this, itemName);
}

void Customer::sendOrder()
{
for (size_t i=0; i<myDemands.size(); i++)
{
  if (myCompany)
    myCompany->receiveDemand(this, myDemands.at(i));
}
myDemands.clear();
}

Item* Warehouse::checkItem(const string& itemToCheck)
{
for (size_t i=0; i<myItems.size(); i++)
{
  Item* item = myItems.at(i);
  if (item->getName() == itemToCheck)
  {
    myItems.erase(myItems.begin()+i);
    return item;
  }
}
return NULL;
}

void Customer::notify(Item* item)
{
Cout << "Customer \"" << name
     << "\" received item \"" << item->getName() <<"\"." << endl;
}

DeliveryPackage* Supplier::receiveDemand(Demand* d)
{
DeliveryPackage *dp = new DeliveryPackage(d);
for (unsigned i=0; i<d->getAmount(); i++)
  dp->putItem(new Item(d->getItemName()));
return dp;
}

void populateItems(Company* c)
{
Warehouse *wh = c->myWarehouse;
wh->addItem (new Item ("USB Flash"));
wh->addItem (new Item ("USB Flash"));
wh->addItem (new Item ("Keyboard"));
wh->addItem (new Item ("Disc"));
wh->addItem (new Item ("Mouse"));
wh->addItem (new Item ("Mouse"));
}
```

```
Customer::~Customer()
{
myDemands.clear();
}

int main()
{
// Create and initialize a company.
Company* company = new Company("Computer Shop");
populateItems(company);

// Create and initialize a customer.
Customer* cust = new Customer("ETF");
cust->setCompany(company);

// Customer creates a list of demands.
cust->createDemand("Disc");
cust->createDemand("Disc");
cust->createDemand("Disc");
cust->createDemand("Mouse");
cust->createDemand("Mouse");
cust->createDemand("Keyboard");
cust->createDemand("Keyboard");

// Customer sends its order.
cust->sendOrder();

// Resource deallocation.
delete cust;
delete company;

return 0;
}
```

2.9. Stabla binarnog pretraživanja

Impementirati na programskom jeziku C++ klasu BinaryTree, koja predstavlja stablo binarnog pretraživanja. Svaki podatak Data u stablu je mapiran na odgovarajući jedinsteni ključ Key.

Klasa BinaryTree treba da bude implementirana u skladu sa sledećim zahtevima:

Stablo se stvara prazno (podrazumevani konstruktor, BinaryTree).

Stablo može da se napravi i kao kopija već postojećeg stabla (konstruktor kopije sa dubokim kopiranjem i konstruktor kopije sa premeštanjem vrednosti, move copy constructor).

Zabraniti dodelu vrednosti jednog stabla drugom stablu.

U stablo se može staviti podatak sa zadatim ključem.

Implementirati javnu metodu bool insert(const Key& key, Data* data)). Metoda vraća true ukoliko je podataka uspešno ubačen u stablo, a false ukoliko se promađe duplikat ključa.

Može se proveriti da li se zadati ključ već nalazi u stablu.

Implementirati javnu metodu bool lookup(const Key& key) const. Metoda vraća true ukoliko se ključ pronađe u stablu, u suprotom false.

Može se dohvatiti podatak na osnovu zadatog ključa.

Implementirati javnu metodu Data* getData(const Key& key) const; Metoda vraća pokazivač na podatak ukoliko se pronađe zadati ključ, u suprotnom nullptr.

Omogućiti propisno uništavanje stabla u destruktoru klase.

Rešenje:

```
#ifndef DATATYPES_H_
#define DATATYPES_H_

#include <iostream>
using namespace std;

// Abstract datatype Key.
class Key
{
unsigned long value;
public:
explicit Key(unsigned long k=0) : value(k) {}

// Compares two keys.
// Returns a negative value if this key is less than the other,
// zero if the keys are equal,
// or a positive value if this key is greater than the other.
int compare (const Key& other) const
{
   return value - other.value;
}

// Overloaded operator for printing Key to an output stream.
friend ostream& operator<<(ostream& out, const Key& key)
{
   return out << key.value;
}
};

// Apstract data wrapper.
class Data {
void* dataPtr;
public:
explicit Data (void* ptr): dataPtr(ptr) {}

void* getData() const { return dataPtr; }
};

#endif /* DATATYPES_H_ */
```

```
#ifndef BINARY_TREE_H_
#define BINARY_TREE_H_

#include "datatypes.h"

class BinaryTree
{
struct TreeNode
{
   Key key;
   const Data *data;
   TreeNode *left;
   TreeNode *right;

   TreeNode(const Key& key, const Data *data)
      : key(key),
        data(data),
        left(nullptr),
        right(nullptr)
   { }
};

// Pointer to the root node.
TreeNode *root;

// Disabled assignment operator.
BinaryTree& operator=(const BinaryTree& right) =delete;

public:
/* ------------------------------------------------------
 * Special methods: constructors, operators and destructor.
 * ------------------------------------------------------ */
// Default constructor: creates an empty binary tree.
BinaryTree(): root(nullptr) {}

// Copy constructor: deep copying of the tree.
BinaryTree(const BinaryTree& tree) { root = hCopyNodes(tree.root); }

// Move copy constructor.
BinaryTree(BinaryTree&& tree) { hMoveNodes(tree); }

// Destructor: removes and deallocates all nodes from the tree.
~BinaryTree() { hClearNodes(root); };

/* ------------------------------------------------------
 * Public interface methods.
 * ------------------------------------------------------ */
// Adds a new value "val" to the tree.
bool insert(const Key& key, const Data* val);

// Check whether given value "val" exists in the tree.
bool lookup (const Key& val) const { return (hSearch(root, val) != nullptr); }

// Retrieves value mapped under the given key.
const Data* get(const Key& val) const { return  hSearch(root, val); }

private:
// Private utility methods.
```

```
// The prefix "h" denotes this method as an internal helper method.

// Clears all nodes from the tree.
void hClearNodes(TreeNode*& node);

// Performs deep copying of tree nodes.
TreeNode* hCopyNodes(const TreeNode* node);

// Moves nodes from the tree designated by "tmpTree.root" to this object.
void hMoveNodes(BinaryTree& tmpTree) { root = tmpTree.root; tmpTree.root = nullptr; }

// Traverses the tree to find value "val".
const Data* hSearch(const TreeNode* node, const Key& key) const;

// Helper recursive method for insertion.
bool hInsert(TreeNode* node, const Key& key, const Data* val);
};

#endif /* BINARY_TREE_H_ */


/* binary_tree.cpp */
#include "binary_tree.h"
#include <iostream>

BinaryTree::TreeNode* BinaryTree::hCopyNodes(const TreeNode* node)
{
if (node == nullptr) return nullptr;

TreeNode* nodeCopy = new TreeNode(node->key, node->data);
nodeCopy->left = hCopyNodes(node->left);
nodeCopy->right = hCopyNodes(node->right);
return nodeCopy;
}

void BinaryTree::hClearNodes(TreeNode*& node)
{
if (node == nullptr)
   return;
if (node->left != nullptr)
   hClearNodes (node->left);
if (node->right != nullptr)
   hClearNodes (node->right);

node = nullptr;
}

const Data* BinaryTree::hSearch(const TreeNode* node, const Key& key) const
{
if (node == nullptr) return nullptr;

if (key.compare(node->key) == 0)
   return node->data;
if (key.compare(node->key) < 0)
   return hSearch(node->left, key);
else
   return hSearch(node->right, key);
}
bool BinaryTree::insert(const Key& key, const Data* val)
```

```
{
if (root == nullptr)
{
   root = new TreeNode(key, val);
   return true;
}
else
{
   return hInsert(root, key, val);
}
}

bool BinaryTree::hInsert(TreeNode* node, const Key& key, const Data* data)
{
if (key.compare(node->key) == 0)
   return false;
// Go to the left subtree.
if (key.compare(node->key) < 0)
{
   if (node->left == nullptr)
      node->left = new TreeNode(key, data);
   else
      hInsert (node->left, key, data);
}
else // Go to the right subtree.
{
   if (node->right == nullptr)
      node->right = new TreeNode(key, data);
   else
      hInsert(node->right, key, data);
}
return true;
}
```

```
#ifndef STUDENT_H_
#define STUDENT_H_

#include "binary_tree.h"

class Student
{
public:
Student (int year, int num)
   : year(year), number(num),
     index(uniqueId(year, num))
{ }

// The unique identifier of a student.
const Key& getIndex() const { return index; }
int getYear() const { return year; }
int getNumber() const { return number; }

private: // Private fields.
int year;
int number;
Key index;
// Generates unique identifier from year and number.
int uniqueId(int year, int num) const { return year * 1000000 + number; };
};

#endif /* STUDENT_H_ */


/* tree_test.cpp*/
#include "binary_tree.h"
#include "student.h"

#include <algorithm>
#include <iostream>
#include <random>
#include <vector>

using namespace std;

vector<Student*>* createTestModel()
{
const unsigned NUMBER_OF_STUDENTS = 10000;
// Create a vector of 10000 sequential int numbers.
vector<unsigned> numbers(NUMBER_OF_STUDENTS);
iota(numbers.begin(), numbers.end(), 1);

// Random shuffling numbers using the built-in random generator.
random_shuffle(numbers.begin(), numbers.end());

// Initial capacity of the vector is desired for efficient execution.
// No more reallocations are necessary.
// Use it whenever the size of vector is known at compile-time.
vector<Student*> *students = new vector<Student*>(NUMBER_OF_STUDENTS);
for (size_t i=0; i<NUMBER_OF_STUDENTS; i++)
   students->at(i) = new Student(2014, numbers[i]);
return students;
}
```

```
/* test.cpp*/

// First user-defined header files.
#include "binary_tree.h"
#include "student.h"

// Then the system headers (notice the alphabetic order!).
#include <algorithm>
#include <cassert>
#include <chrono>
#include <iostream>
#include <random>
#include <vector>

using namespace std;
using namespace std::chrono;

// A helper function that compares two students by index.
bool order_asc(Student* s1, Student* s2)
{
return s1->getIndex().compare(s2->getIndex()) < 0;
}

void testStudents()
{
BinaryTree* tree = new BinaryTree();
// Create a vector (linear structure) of students.
vector<Student*>* students = createTestModel();

// The variable "it" represents an object that allows
// the sequential access to the elements of a collection
// (iterates through the collection).
// In this loop, "vector<Student*>::iterator it" is
// defined by specifying its type explicitly.
// Instead of explicit type specifying, one can use the keyword "auto".
// In that case, the compiler performs type deduction from the context.
// auto it=students->begin();  <=> vector<Student*>::iterator it=students->begin();
for (vector<Student*>::iterator it = students->begin();
     it != students->end(); ++it)
{
   Student* s = *it;
   if (!tree->insert(s->getIndex(), new Data(s)))
   {
      cout << "Student "
         << s->getYear() << "/" << s->getNumber()
         << " already exists!" << endl;
   }
}

// Try to put a duplicate student into the tree.
Student* duplicate = students->at(0);
Data* duplicateData = new Data(duplicate);
if (!tree->insert(duplicate->getIndex(), duplicateData))
{
   cout << "** Student "
   << duplicate->getYear() << "/" << duplicate->getNumber()
   << " already exists!" << endl;
   delete duplicateData;
   duplicateData = nullptr;
}
```

```
// Generate an array of random indexes for search testing.
const unsigned NUM_TEST_STUDENTS = 5000;
vector<unsigned> randomIndexes(NUM_TEST_STUDENTS);

// Seed with a real random value, if available.
random_device rd;
default_random_engine e1(rd());

// Choose a random mean between 1 and 10000
uniform_int_distribution<int> uniform_dist(1, 9999);

for (size_t i=0; i<5000; i++)
{
   randomIndexes[i] = uniform_dist(e1);
}

// Demonstrate sequential search in an unsorted collection.
cout << "Sequential search (unsorted array): ";
int numFound = 0;
auto begin = high_resolution_clock::now();
for (auto idxIt = randomIndexes.begin(); idxIt != randomIndexes.end(); ++idxIt)
{
   unsigned randomIdx = *idxIt;
   Student* studentToSearch = students->at(randomIdx);
   for (auto studentIt = students->begin(); studentIt != students->end(); ++studentIt)
   {
      Student* current = *studentIt;
      if (current->getIndex().compare(studentToSearch->getIndex()) == 0)
      {
         numFound++;
         break;
      }
   }
}
auto end = high_resolution_clock::now();
cout << "done in " << duration_cast<nanoseconds>(end-begin).count() << "ns" << endl;
assert(numFound == NUM_TEST_STUDENTS);

// ------------
// Demonstrate random search with the binary tree.
cout << "Search with binary tree: ";
numFound = 0;
begin = high_resolution_clock::now();

for (auto rndIt = randomIndexes.begin(); rndIt != randomIndexes.end(); ++rndIt)
{
   Student* student = students->at(*rndIt);
   if (tree->lookup(student->getIndex()))
      numFound++;
}

end = high_resolution_clock::now();
cout << "done in " << duration_cast<nanoseconds>(end-begin).count() << "ns" << endl;
assert(numFound == NUM_TEST_STUDENTS);

// ----------
// Demonstrate sequential search in an unsorted collection.
```

```
// Sort students by index in ascending order.
// sort(students->begin(), students->end(), STUDENT_COMPARATOR_ASC);
sort(students->begin(), students->end(), order_asc);

cout << "Sequential search (sorted array): ";
numFound = 0;
begin = high_resolution_clock::now();
for (auto idxIt = randomIndexes.begin(); idxIt != randomIndexes.end(); ++idxIt)
{
   unsigned randomIdx = *idxIt;
   Student* studentToSearch = students->at(randomIdx);
   for (auto studentIt = students->begin(); studentIt != students->end(); ++studentIt)
   {
      Student* current = *studentIt;
      if (current->getIndex().compare(studentToSearch->getIndex()) == 0)
      {
         numFound++;
         break;
      }
   }
}

end = high_resolution_clock::now();
cout << "done in " << duration_cast<nanoseconds>(end-begin).count() << "ns" << endl;
assert(numFound == NUM_TEST_STUDENTS);

cout << "Binary search: ";
numFound = 0;
begin = high_resolution_clock::now();

for (auto idxIt = randomIndexes.begin(); idxIt != randomIndexes.end(); ++idxIt)
{
   Student* studentToSearch = students->at(*idxIt);
   if (binary_search(students->begin(), students->end(), studentToSearch, order_asc))
   {
      numFound++;
   }
}

end = high_resolution_clock::now();
cout << "done in " << duration_cast<nanoseconds>(end-begin).count() << "ns" << endl;
assert(numFound == NUM_TEST_STUDENTS);

// Print all students to the standard output.
for (auto studentIt = students->begin(); studentIt != students->end(); ++studentIt)
{
   cout << "Student " << (*studentIt)->getIndex() << endl;
}

// Deallocate memory.
for (auto it = students->begin(); it != students->end(); ++it)
   delete *it;
delete students;
delete tree;
}
```

```cpp
/*
 * main.cpp
 *
 *  Created on: 04.11.2014.
 *      Author: Nemanja Kojic
 */

#include "binary_tree.h"
#include "datatypes.h"
#include "student.h"

// Declaration of the function that tests BinaryTree class.
void testStudents();

int main()
{
testStudents();
}
```

2.10. Obilazak (binarnog) stabla

Na programskom jeziku C++ dopuniti implementaciju klase BinaryTree omogućavanjem proizvoljnog obilaska čvorova. Obilazak stabla (*Traversal*) ima zadatak da obiđe čvorove stabla i da sa svakim čvorom obavi definisanu obradu. Klasa Traversal treba da implementira 3 tipa obilaska (preorder, inorder, postorder), tako da se za svaki tip obilaska ponudi i rekurzivna i iterativna implementacija.

Klasu traversal implementirati prema sledećim zahtevima:

Objekat klase Traversal se stvara sa zadatim načinom obilaska (podrazumevano rekurzivni preorder).

Implementirati konstruktor klase.

Omogućiti obilazak zadatog stabla.

Implementirati metodu **`void traverse(BinaryTree*)`** koja vrši obilazak zadatog stabla. Obilazak stabla je definisan prilikom pravljenja objekta klase Traversal.

Omogućiti sve tipove obilaska.

Implementirati pomoćne metode za pojedinačne tipove obilaska. Primer prototipa metode je:

`void recursivePreorder(BinaryTree::TreeNode*);`

Klasu BinaryTree dopuniti u skladu sa sledećim zahtevima:

Omogućiti proizvoljan obilazak stabla.

Implementirati metodu void traverse(Traversal& t) koja prihvata zadatu strategiju obilaska i delegira joj obilazak.

Rešenje:

U klasi stabla su navedene samo one deklaracije/definicije koje su dodate za potrebe obilaska. Pored toga, neki članovi su ostavljeni i kao podsetnik na celokupnu strukturu klase (TreeNode, root). Rešenje još uvek nije potpuno objektno orijentisano, zbog nedostataka koncepata polimorfizma i nasleđivanja!

```
// Forward declaration of the class Traversal (needed for the method traverse).
class Traversal;

class BinaryTree
{
class TreeNode { /*...*/ }
// Pointer to the root node.
TreeNode *root;

// The friend class Traversal.
// The class must be declared as a friend of BinaryTree,
// in order to be able to access the private class TreeNode (and its members).
friend class Traversal;

public:
// ... constructors and other public methods.

// Traverses the tree.
void traverse(Traversal& t) { t.traverse(this); };

private:
// ... (other private helper methods)
};
```

```
/*
 * traversal.h
 *
 *  Created on: 18.11.2014.
 *      Author: Nemanja Kojic
 */
#ifndef TRAVERSAL_CPP_
#define TRAVERSAL_CPP_

#include "binary_tree.h"

class Traversal
{
public:

// Available traversal algorithms.
enum TraversalKind
{
   PREORDER_RECURSIVE,
   INORDER_RECURSIVE,
   POSTORDER_RECURSIVE,
   PREORDER_ITERATIVE,
   INORDER_ITERATIVE,
   POSTORDER_ITERATIVE,
};

Traversal (TraversalKind tk = TraversalKind::PREORDER_RECURSIVE)
: selectedTraversal (tk)
{}

// Public interface method: invoked from within BinaryTree class.
// Traverses the binary tree.
void traverse(BinaryTree* tree);

private:

void visitNode(BinaryTree::TreeNode* t);

void recursivePreorder(BinaryTree::TreeNode* t);
void iterativePreorder(BinaryTree::TreeNode* t);

void recursiveInorder(BinaryTree::TreeNode* t);
void iterativeInorder(BinaryTree::TreeNode* t) { /* homework */ }

void recursivePostorder(BinaryTree::TreeNode* t);
void iterativePostorder(BinaryTree::TreeNode* t);

TraversalKind selectedTraversal;
};

#endif /* TRAVERSAL_CPP_ */
```

```
/*
 * traversal.cpp
 *
 *  Created on: 18.11.2014.
 *      Author: Nemanja Kojic
 */

#include "traversal.h"

#include <cassert>
#include <stack>

using namespace std;

void Traversal::recursivePreorder(BinaryTree::TreeNode* node)
{
if (node == nullptr)
   return;

visitNode(node);
recursivePreorder(node->left);
recursivePreorder(node->right);
}

void Traversal::iterativePostorder(BinaryTree::TreeNode* node)
{
if (node == nullptr)
   return;
BinaryTree::TreeNode* lastVisitedNode = nullptr;
stack<BinaryTree::TreeNode*>* unvisitedNodes = new stack<BinaryTree::TreeNode*>;
while (!unvisitedNodes->empty() || node != nullptr)
{
   if (node != nullptr)
   {
      unvisitedNodes->push(node);
      node = node->left;
   }
   else
   {
      BinaryTree::TreeNode* topNode = unvisitedNodes->top();
      if (topNode->right != nullptr && lastVisitedNode != topNode->right)
      {
         node = topNode->right;
      }
      else
      {
         visitNode(topNode);
         lastVisitedNode = topNode;
         unvisitedNodes->pop();
      }
   }
}
bool noUnvisitiedNodes = unvisitedNodes->empty();
delete unvisitedNodes;
assert (noUnvisitiedNodes);
}
```

```
void Traversal::iterativePreorder(BinaryTree::TreeNode* node)
{
if (node == nullptr)
   return;

stack<BinaryTree::TreeNode*>* unvisitedNodes = new stack<BinaryTree::TreeNode*>;
while (!unvisitedNodes->empty() || node != nullptr)
{
   if (node != nullptr)
   {
      // Visit current node.
      visitNode(node);

      // Save the backtracking position.
      if (node->right != nullptr)
         unvisitedNodes->push(node->right);

      // Move to the left subtree.
      node = node->left;
   }
   else
   {
      // Restore the nearest backtracking position.
      node = unvisitedNodes->top();
      unvisitedNodes->pop();
   }
}
bool noUnvisitiedNodes = unvisitedNodes->empty();
delete unvisitedNodes;
assert (noUnvisitiedNodes);
}

void Traversal::recursiveInorder(BinaryTree::TreeNode* node)
{
if (node == nullptr)
   return;

// Recursive inorder traversal.
recursiveInorder(node->left);
visitNode(node);
recursiveInorder(node->right);
}

void Traversal::recursivePostorder(BinaryTree::TreeNode* node)
{
if (node == nullptr)
   return;

// Recursive postorder traversal.
recursivePostorder(node->left);
recursivePostorder(node->right);
visitNode(node);
}
```

```
void Traversal::visitNode(BinaryTree::TreeNode* t)
{
cout << t->key << ",";
}

void Traversal::traverse(BinaryTree* tree)
{
switch (selectedTraversal)
{
   case PREORDER_RECURSIVE:
      recursivePreorder(tree->root);
      break;
   case PREORDER_ITERATIVE:
      iterativePreorder(tree->root);
      break;
   case INORDER_RECURSIVE:
      recursiveInorder(tree->root);
      break;
   case INORDER_ITERATIVE:
      iterativeInorder(tree->root);
      break;
   case POSTORDER_RECURSIVE:
      recursivePostorder(tree->root);
      break;
   case POSTORDER_ITERATIVE:
      iterativePostorder(tree->root);
      break;
   default:
      iterativeInorder(tree->root);
      break;
}
}
```

VAŽNA NAPOMENA!

Metoda Traversal::traverse nije implementirana prema principima OO paradigme. Svaka pojava if-else ili switch-case strukture, sa navođenjem konkretnih tipova u OO programu ukazuje na nedostatke u dizajnu OO rešenja. OO program treba da bude što nezavisniji od konkretnih tipova podataka (klasa) i da se po mogućstvu oslanja samo na apstraktne tipove podata i interfejse.

Međutim, u ovom slučaju, metoda je implementirana na ovaj način jer koncepti polimorfizma i nasleđivanja nisu još uvek poznati, tako da je rešenje trenutno više objektno bazirano, nego objektno orijentisano!

```
/*
 * test.cpp
 *
 *  Created on: 12.11.2014.
 *      Author: Nemanja Kojic
 */

#include "binary_tree.h"
#include "traversal.h"

void testTraversal()
{
BinaryTree* tree = new BinaryTree();
tree->insert(Key(10), nullptr);
{
   tree->insert(Key(5), nullptr);
   {
      tree->insert(Key(3), nullptr);
      tree->insert(Key(7), nullptr);
   }
   tree->insert(Key(15), nullptr);
   {
      tree->insert(Key(12), nullptr);
      tree->insert(Key(20), nullptr);
   }
}

cout << "Recursive PREORDER: ";
Traversal tpr(Traversal::PREORDER_RECURSIVE);
tree->traverse(tpr);
cout << endl;

cout << "Iterative PREORDER: ";
Traversal tpi(Traversal::PREORDER_ITERATIVE);
tree->traverse(tpi);
cout << endl;

cout << "Recursive INORDER: ";
Traversal trin(Traversal::INORDER_RECURSIVE);
tree->traverse(trin);
cout << endl;

cout << "Recursive POSTORDER: ";
Traversal trpost(Traversal::POSTORDER_RECURSIVE);
tree->traverse(trpost);
cout << endl;

cout << "Iterative POSTORDER: ";
Traversal tpost(Traversal::POSTORDER_ITERATIVE);
tree->traverse(tpost);
cout << endl;
}
```

3. Rad sa ulaznim i izlaznim tokovima podataka

3.1. Zadatak

Napisati program na programskom jeziku C++ koji ilustruje rad sa ulaznim datotekama i prikazuje korišćenje indikatora stanja toka.

Rešenje:

Sadržaj ulazne datoteke je:

```
jedan dva tri cetiri\n
21 pet sest sedam\n
31 32 osam devet deset\n
41\n
```

```
// za rad sa fajlovima
#include<fstream>
// za rad sa glavnim izlazom
#include<iostream>

using namespace std;

void printStreamStatus(const iostream& dat1) {
    cout<<"(b,g,f,e)=" <<dat1.bad()<<dat1.good()<<dat1.fail()<<dat1.eof()<< endl;
}

void main() {
const int maxLineLength = 100;
char line[maxLineLength];

// stvori objekat tipa fstream i otvari pridruzeni fajl za citanje
fstream dat1("ulaz.txt", ios::in);
printStreamStatus(dat1); // 0100/0100

// dok ne stigne do kraja fajla
// peek() bez promene stanja toka vraca znak koji bi bio procitan prilikom
    // sledeceg citanja iz toka
while(dat1.peek() != EOF)
//   eof() kaze da li je prilikom poslednjeg citanja procitan EOF
//NAPOMENA: posmatrati ponasanje programa u zavisnosti od uslova za while
//while(!dat1.eof()) // jedna iteracija vise!
{
   // citaj liniju po liniju
   dat1.getline(line, maxLineLength);
   printStreamStatus(dat1); // 0100/ (0100 ili 0011 u poslednjoj iteraciji)
   // i ispisuj na glavnom izlazu
   cout << line << "\n";
}
// po obavljenom citanju, zatvori fajl
printStreamStatus(dat1); // 0001/0011
dat1.close();
cout << "Posle zatvaranja" << endl;
printStreamStatus(dat1); 0001/0011

fstream nema("nema.txt", ios::in);
printStreamStatus(nema); 0010
}
```

3.2. Zadatak

Napisati program koji ilustruje rad sa izlaznim tokovima, prenos podataka sa i bez konverzije.

Rešenje:

Sadržaj ulazne datoteke je:

```
jedan dva tri cetiri\n
21 pet sest sedam\n
31 32 osam devet deset\n
41\n
```

```
#include<fstream>
#include<iostream>
#include<iomanip>
using namespace std;

void mai()
{
fstream dat1("ulaz.txt", ios::in);

const streamsize MAX_DUZ = 80;
char line[MAX_DUZ];
int i, pozicija;
string rec = "Perica";
cout << rec.c_str() << endl;

// tellg vraca poziciju unutar ulaznog toka
// tellp vraca poziciju unutar izlaznog toka (ne koristi se u ovom zadatku)
pozicija = dat1.tellg();
cout << "Pozicija: " << pozicija << endl;

// prva linija datoteke: "jedan dva tri cetiri"
dat1 >> line; // citamo prvu rec
cout << line << endl; // "jedan"

dat1.width(3);
dat1 >> line; // citamo 2 znaka sledece reci ili jednu rec ne duzu od dva znaka
cout << line << endl; // "dv"

dat1.width(3);
dat1 >> line; // citamo 2 znaka sledece reci ili jednu rec ne duzu od dva znaka
cout << line << endl; // "a"

dat1.width(3);
dat1 >> line; // citamo 2 znaka sledece reci ili jednu rec ne duzu od dva znaka
cout << line << endl; // "tr"

dat1.getline(line, MAX_DUZ); // citamo sve do kraja linije;
// po zelji, mozemo kao treci argument metode getline zadati znak
// koji ce sluziti kao delimiter sa kojim se zavrsava citanje
cout << line << endl; // "i cetiri"

pozicija = dat1.tellg();
cout << "Pozicija: " << pozicija << endl;

dat1 >> i >> line; // citamo int, pa jednu rec
cout << i << line << endl; // "21pet"
dat1.getline(line, MAX_DUZ); // citamo sve do kraja linije
cout << line << endl; // " sest sedam"
```

```
pozicija = dat1.tellg();
cout << "Pozicija: " << pozicija << endl;

// pomeramo se 10 mesta unapred u odnosu na trenutnu poziciju
dat1.seekg(10, ios::cur);

pozicija = dat1.tellg();
cout << "Pozicija: " << pozicija << endl;

// ispis istih brojeva, samo sa razlicitim brojem decimalnih cifara i slovnih mesta
cout.width(12);
cout << pozicija << endl; // "          51"

cout.width(12);  // width se odnosi samo na prvi naredni podatak koji se ispisuje
// "  Pozicija: 51"
cout << "Pozicija: " << pozicija << endl;

// setw je deklarisan u <iomanip> i kao i width, utice samo na prvi naredni podatak
// "  Pozicija:    51"
cout << setw(12) << "Pozicija: " << setw(5) << pozicija << endl;

cout.fill('-'); // fill odredjuje cime ce biti popunjena prazna mesta u ispisu
cout.width(10);
cout << pozicija <<endl; // "--------51"
cout << pozicija << endl; // ponisteno dejstov manipulatora: "51"

cout.width(5);
cout.fill('+');
cout << pozicija << endl; // "+++51"

// manipulator setfill ima istu ulogu kao metoda fill
cout << setw(4) << setfill ('0') << pozicija << endl; // "0051"

long coutFlags = cout.flags(); // pamtimo polazno stanje podesavanja za formatiranje

// "Pozicija kao double: 51"
cout << "Pozicija kao double: " << (double) pozicija << endl;

// broj znacajnih cifara za prikaz, odnosi se na svaki naredni ispis realnih brojeva
cout.precision(4);
cout.flags(coutFlags | ios::showpoint); // forsiramo pojavljivanje decimalne tacke

// "Pozicija kao double: 51.00"
cout << "Pozicija kao double: " << (double) pozicija << endl;

// manipulator setprecision ima istu ulogu kao metoda precision
// "Pozicija kao double: 51.0000"
cout << "Pozicija kao double: " << setprecision(6) << (double) pozicija << endl;

// sada se precision odnosi na broj cifara iza decimalne tacke
cout.flags(coutFlags | ios::fixed);

// "Pozicija kao double: 51.000000"
cout << "Pozicija kao double: " << (double) pozicija << endl;

// scientific prikaz, precision i dalje odredjuje broj cifara iza decimalne tacke
cout.flags(coutFlags | ios::scientific);
// "Pozicija kao double: 5.100000e+001"
cout << "Pozicija kao double: " << (double) pozicija << endl;
```

```
  // iskljucujemo scientific prikaz (resetovana preciznost)
  cout.flags(cout.flags() & ~ios::scientific);

  // "Pozicija kao double: 51"
  cout << "Pozicija kao double: " << (double) pozicija << endl;

  // vracamo format na polazno stanje
  cout.flags(coutFlags);


  // pomeramo se 0 mesta u odnosu na pocetak toka (ios::end za pomeraj u odnosu na kraj
toka)
  dat1.seekg(0, ios::beg);

  pozicija = dat1.tellg();
  cout << "Pozicija: " << pozicija << endl;

  // ispisujemo sadrzaj datoteke na standardnom izlazu
  while(!dat1.eof())
  {
    dat1.getline(line, MAX_DUZ);
    if ('\0' != line[0])
      cout << line << "\n";
  }
  dat1.close();
  }
```

4. Preklapanje operatora

4.1. Kompleksni brojevi (operator za konverziju tipa i operator+).

Napisati na programskom jeziku C++ klasu kompleksnih brojeva. Kompleksni broj se stvara zadavanjem realnog i imaginarnog dela. Ukoliko se za neki deo eksplicitno ne zada vrednost podrazumeva se nula. Preklopiti operator za konverziju tipa koji konvertuje kompleksni broj u njegov moduo i preklopiti operator za sabiranje tako da se omogući sabiranje kompleksnih brojeva. Primeniti strogu enkapsulaciju podataka.

Rešenje:

```
// Definisanje konverzija tipova.

#include <iostream>
#include <cmath>
using namespace std;

class Kompl {
  double re, im;
public:
  Kompl (double r=0, double i=0) { re=r; im=i; } // Inic. ili konverzija.
  operator double() { return sqrt(re*re+im*im); } // Konverzija u double.
  friend double real (Kompl z) { return z.re; }  // Realni deo.
  friend double imag (Kompl z) { return z.im; }  // Imaginarni deo.
  Kompl operator+ (Kompl z)                      // Sabiranje.
    { z.re += re; z.im += im; return z; }

/*friend Kompl operator+(const Kompl& c1, const Kompl& c2) {
   return Kompl(c1.re+c2.re, c1.im+c2.im);
    }*/
//void operator@(Kompl k) {}
} ;


void pisi (const char* c, Kompl z)
  { cout << c << " = (" << real (z) << ',' << imag (z) << ")\n"; }

int main () {
  Kompl a (1, 2);      pisi ("a      ", a);
  Kompl b = a;         pisi ("b      ", b);
  Kompl c = 5;         pisi ("c      ", c);
                       pisi ("a+b    ", a + b);
                       pisi ("a+3    ", a + (Kompl)3);
                       // Sta se desava ako nema eksplicitne konverzije?
                    // Moze i Kompl(3)

                    pisi ("|a|+3  ", (double) a + 3); // <=> a.operator double()+3
                       pisi ("a+(3,4)", a + Kompl (3,4));
                       cout << "dble(a) = " << (double)a << endl;

  double d=Kompl(3,4); cout << "d       = " << d << endl;
// Vrsi se implicitna konverzija iz Kompl;

  cout << "cplx=    " << b << endl;  // Ne postoji preklapanje operator <<.
                       // Zbog toga se pokusa konverzija u skalarni tip.
                    // Postoji (double) koje se i primenjuje u ovom slucaju.
}
```

Komentar:

Operator + može da se preklopi kao metoda ili kao prijateljska funkcija. Ako se preklopi kao metoda, a ne kao prijateljska funkcija, ne primenjuje se implicitna konverzija na levi operand (operandi nisu simetrični).

4.2. Tekst (operator=).

Proširiti klasu **Tekst** napisanu na programskom jeziku C++ preklapanjem operatora za dodelu vrednosti, tako da se omogući dodela znakovnog niza objektu sa leve strane operatora dodele iz objekta klase **Tekst** na desnoj strani operatora dodele. Operatorska funkcija za dodelu vrednosti mora biti bezbedna.

Rešenje:

```
// Inicijalizacija i dodela vrednosti.
#include <cstring>
#include <iostream>
using namespace std;

class Text {
  char* txt;                              // Pokazivac na niz znakova.
public:

  Text (const char* string) {               // Konverzija iz char* u Text.
    txt = new char [strlen(string)+1];
    strcpy (txt, string);
  }

  Text (const Text& other) {              // Inicijalizacija objektom klase Text.
    txt = new char [strlen(other.txt)+1];
    strcpy (txt, other.txt);
  }

  Text& operator= (const Text& other) {  // Dodela vrednosti.
    if (this != &other) {
      delete [] txt;
      txt = new char [strlen(other.txt)+1];
      strcpy (txt, other.txt);
    }
    return *this;
  }

  ~Text () { cout<<"Pozvan destruktor za " <<txt<<endl;  delete [] txt; }
};

int main () {
  Text a("Dobar dan.");                 // Stvaranje i inicijalizacija.
  Text b = Tekst ("Zdravo.");
  Text c(a), d = b;    // Treba izbegavati drugi vid inicijalizacije operatorom =
  a = b;                                  // Dodela vrednosti.
}
```

Komentar:

Kod dodele vrednosti, za razliku od inicijalizacije, smatra se da oba objekta operanda već postoje i imaju definisano stanje. Operator za dodelu se preklapa tako što se prvo uništi stanje objektu sa leve strane, a zatim se kopira stanje iz objekta sa leve strane.

Šta ako se napiše sledeći izraz: a = a ? Postoji isti operand sa leve i desne strane. Uništavanje stanja levom operandu, zapravo znači uništavanje stanja desnom operandu. Ovo je specijalan slučaj i rešava se sledećim uslovnim izrazom: `if (this != &tks) { ... }`.

Ovaj uslov je tačan samo kad levi i desni operand predstavljaju isti objekat i tada operatorska funkcija ne treba da radi ništa!

4.3. Kompleksni brojevi (kompletan primer).

Ilustrovati pravila i preporuke za preklapanje operatora jezika C++ na primeru klase kompleksnih brojeva **`Complex`**.

Rešenje:

```
// Primer 10: Klasa kompleksnih brojeva
// File: complex.h

#ifndef _Complex
#define _Complex

#include <iostream>
using namespace std;

class Complex {
public:
          Complex (double real=0, double imag=0);
    // Complex (const Complex&);  // NK: Konstruktor kopije nepotreban.
  friend Complex  operator+  (const Complex&, const Complex&);
  friend Complex  operator-  (const Complex&, const Complex&);
  friend Complex  operator*  (const Complex&, const Complex&);
  friend Complex  operator/  (const Complex&, const Complex&);

    // NK: operator dodele vrednosti (obavezno kao metoda)
    // Operator dodele vrednosti nije potreban.
    // Complex& operator=  (const Complex&);

// Operatori koji menjaju stanje objekta za koji su pozvani relizuju se kao metode).
          Complex& operator+= (const double);
          Complex& operator-= (const double);
          Complex& operator*= (const double);
          Complex& operator/= (const double);
          Complex& operator+= (const Complex&);
          Complex& operator-= (const Complex&);
          Complex& operator*= (const Complex&);
          Complex& operator/= (const Complex&);

          //Complex  operator+  () const;
          Complex  operator-  () const;         // Unarni - (kao funkcija clanica).

  friend Complex  operator+  (const Complex&); // Unarni + kao prijateljska funkcija).

  friend int      operator== (const Complex&, const Complex&);
  friend int      operator!= (const Complex&, const Complex&);

  friend double   re         (const Complex&);
  friend double   im         (const Complex&);
  friend Complex  conj       (const Complex&);

  friend ostream& operator<< (ostream&, const Complex&);
  friend istream& operator>> (istream&, Complex&);

  // napisati operator za stepenovanje kompleksnog broja

private:
  double real,imag;
};
#endif
```

```
// Inline functions:

//inline Complex& Complex::operator =(const Complex& c1) {
//   this->real = c1.real; this->imag = c1.imag;
//   return *this;
//}

//inline Complex::Complex(const Complex& c):real(c.real),imag(c.imag) {}

inline Complex::Complex (double r, double i) : real(r), imag(i) {
cout << "Pozvan je konstruktor za objekat " << *this << endl;
}

inline Complex  operator+  (const Complex& c1, const Complex& c2)
  { return Complex(c1.real+c2.real,c1.imag+c2.imag); }

inline Complex  operator-  (const Complex& c1, const Complex& c2)
  { return Complex(c1.real-c2.real,c1.imag-c2.imag); }

inline Complex& Complex::operator+= (const double d)
  { real+=d; return *this; }

inline Complex& Complex::operator-= (const double d)
  { real-=d; return *this; }

inline Complex& Complex::operator*= (const double d)
  { real*=d; imag*=imag; return *this; }

inline Complex& Complex::operator/= (const double d)
  { real/=d; imag/=imag; return *this; }

inline Complex& Complex::operator+= (const Complex &c)
  { real+=c.real; imag+=c.imag; return *this; }

inline Complex& Complex::operator-= (const Complex &c)
  { real-=c.real; imag-=c.imag; return *this; }

inline Complex& Complex::operator*= (const Complex &c)
  { return *this=*this*c; }

inline Complex& Complex::operator/= (const Complex &c)
  { return *this=*this/c; }

//inline Complex  Complex::operator+ () const { return *this; }
inline Complex  Complex::operator- () const { return Complex(-real,-imag); }

inline Complex operator+ (const Complex& c) { return Complex(c); }

inline int  operator== (const Complex& c1, const Complex& c2)
  { return (c1.real==c2.real) && (c1.imag==c2.imag); }

inline int  operator!= (const Complex& c1, const Complex& c2)
  { return !(c1==c2); }

inline double re (const Complex& c) { return c.real; }
inline double im (const Complex& c) { return c.imag; }
inline Complex conj (const Complex& c) { return Complex(-c.real,c.imag); }
```

```
// Primer 10: Klasa kompleksnih brojeva
// File: complex.cpp

#include "complex.h"

Complex  operator*  (const Complex& c1, const Complex& c2)
  { return Complex(c1.real*c2.real-c1.imag*c2.imag,
                   c1.real*c2.imag+c1.imag*c2.real); }

Complex  operator/  (const Complex& c1, const Complex& c2)
  { double r=c2.real*c2.real-c2.imag*c2.imag;
    return Complex((c1.real*c2.real+c1.imag*c2.imag)/r,
                   (c2.real*c1.imag-c2.imag*c1.real)/r); }

ostream& operator<< (ostream &os, const Complex &c)
  { return os<<"("<<c.real<<","<<c.imag<<")"; }

istream& operator>> (istream &is, Complex &c) {
cout << "Realni deo: ";
is >> c.real;
cout << "Imaginarni deo: ";
is>>c.imag;
return is;
}

void main() {
Complex c1(1,2);
Complex c2 = c1;
Complex c3(4,5);
cout << ((+c1)+c2) << endl;
Complex c4;
cout<< (c4 += (c1+c2))*c3 << endl;
cout << "Unos kompleksnog broja:" << endl;
cin >> c4;
cout << "Uneli ste kompleksni broj "<< c4 << endl;
}
```

4.4. Nizovi (preklapanje operatora za indeksiranje, operator[]).

Ilustrovati preklapanje operatora za indeksiranje na primer klase nizova.

Rešenje:

```
  // Preklapanje operatora [].

  #include <iostream>
  #include <cstdlib>
  using namespace std;

  class Niz {
    int poc, kraj;                    // Opseg indeksa.
    double* a;                        // Elementi niza.
    void kopiraj (const Niz&);        // Kopiranje.
  public:
    Niz (int min, int max) {          // Konstruktori.
      if (min > max) exit (1);
      a = new double [(kraj=max)-(poc=min)+1];
    }
    Niz (const Niz& niz) { kopiraj (niz); }  // Konstruktor
    ~Niz () { delete [] a; }          // Destruktor.

    Niz& operator= (const Niz& niz) {  // Dodela vrednosti.
      if (this != &niz) { delete [] a; kopiraj (niz); }
      return *this;
    }
    double& operator[] (int i) {       // DOHVATANJE ELEMENTA.
    cout << "Pozvana regularna metoda " << i<<endl;
      if (i<poc || i>kraj) exit (2);     // Ovo nije dobar primer oporavka od greske.
izuzeci...
      return a[i-poc];
    }
    const double& operator[] (int i) const {
    cout << "Pozvana const metoda "<<i<<endl;
      if (i<poc || i>kraj) exit (2);
      return a[i-poc];
    }
  };

  void Niz::kopiraj (const Niz& niz) { // Kopiranje.
    a = new double [(kraj=niz.kraj)-(poc=niz.poc)+1];
    for (int i=0; i<kraj-poc+1; i++) a[i] = niz.a[i];
  }

  int main () {
    Niz niz (-5, 5);
    for (int i=-5; i<=5; i++) niz[i]=i;
    //*(niz+4) = 4;       // GRESKA: ne vazi adresna aritmetika!
    const Niz cniz = niz;
    //cniz = niz;         // GRESKA: menjanje celog nepromenljivog niza!
    //cniz[2] = 55;       // GRESKA: menjanje dela nepromenljivog niza!
    double a = cniz[2]; // U redu: uzimanje je dozvoljeno.

    // PRIMERI NAREDBI SA NEKONSTANTNIM OBJEKTOM NIZA.
    niz[3] = 1;
    cout << "niz[3] = " << (niz[3]+=1) << endl;

}
```

4.5. Funkcije (preklapanje operatora za poziv funkcije, operator()).

Ilustrovati preklapanje **`operatora()`** na primeru klase za predstavljanje instanci funkcije sinus. Svaka instanca se razlikuje prema parametrima funkcije. Omogućiti pozivom operatorske funkcije **`operator()`** izračunavanje vrednosti funkcije za zadatu vrednost.

Rešenje:

```
// Preklapanje operatora ().

#include <cmath>
#include <iostream>
#include <iomanip>
using namespace std;

// Sprecava narusavanje
class Sin {
  double a, omega, fi;
public:
  explicit Sin (double a=1, double omega=1, double fi=0) {
    this->a = a; this->omega = omega; this->fi = fi;
  }
  double operator() (double x)
    { return a * sin (omega*x+fi); }

  // Za konstantne objekte?
  double operator() (double x) const
  { cout<<"Pozvana const operatorska metoda"<<endl;  return a * sin (omega*x+fi);  }
};

const double PI = 3.141592;

int main () {
  const Sin sin (2, 0.5, PI/6);
  cout << fixed << setprecision(5);
  for (double x=0; x<=2*PI; x+=PI/12)
    cout << setw(10) << x
         << setw(10) << sin(x)
         << setw(10) << std::sin(x) << endl;
}
```

4.6. Clock (preklapanje operatora ++ i --).

Ilustrovati preklapanje operatora ++ i -- na primeru klase časovnika **`Clock`**. Klasa **`Clock`** čuva podatke u satima i minutima, može da primi jedan otkucaj, da ispiše vreme na zadati izlaza i stvara se zadavanjem sata i minuta (podrazumevano 12h i 00min). Preklopiti **`operator++`** koji pomera časovnik za jedan minut unapred. Preklopiti **`operator--`** koji vraća časovnik jedan minut unazad.

Rešenje:

```
#include <iostream>

using namespace std;

class Clock {
public: // Methods
Clock(int h=12, int m=0);     // noon by default
Clock& operator++();          // prefix form; turn the clock 1min forward.
Clock operator++(int);        // postfix form; turn the clock 1min forward.

Clock& operator--();          // prefix form; turn the clock 1min back by.
Clock operator--(int);        // postfix form; turn the clock 1min back by.

private: // Methods
friend ostream& operator<<(ostream& out, Clock& c);
void oneMinuteBack();
void oneMinuteForward();
void rewindClock(int minuteStep) {}  // homework?

private: // Fields
int hour, min;

public: // Constants
static const int MIN_PER_HOUR=60;
static const int HOUR_PER_DAY=24;
};

Clock::Clock(int h, int m) : hour(h), min(m) { }

void Clock::oneMinuteBack() {
if (--min < 0) min = MIN_PER_HOUR-1;
if (min == MIN_PER_HOUR-1) {
   if (--hour < 0) hour=HOUR_PER_DAY-1;
}
}

void Clock::oneMinuteForward()
{
min = (min + 1) % MIN_PER_HOUR;
if (min == 0)
   hour = (hour + 1) % HOUR_PER_DAY;
}

ostream& operator<<(ostream& os, Clock& c)
{
if (c.hour < 10) os << '0';  // then pad the hour with a 0
os << c.hour << ":";
if (c.min < 10) os << '0';  // then pad the min with a 0
os << c.min;
return os;
}
```

```
Clock& Clock::operator++()    // prefix form
{
oneMinuteForward();             // increment
return *this;       // return itself for assignment
}

Clock Clock::operator++(int)   // parameter ignored and does
{                              // not even need to be named.
Clock c = *this;  // Save the object.
oneMinuteForward();           // Increment the object.
return c;         // Return the object as it was before incrementing.
}

Clock& Clock::operator--()    // prefix form
{
oneMinuteBack();             // decrement
return *this;       // return itself for assignment
}

Clock Clock::operator--(int)   // parameter ignored and does
{                              // not even need to be named.
Clock c = *this;  // Save the object.
oneMinuteBack();           // Decrement the object.
return c;         // Return the object as it was before incrementing.
}


int main()
{
Clock c1, c2(3,20), c3(10,59);

cout << c1 << " " << c2 << " " << c3 << endl;
// should output 12:00 03:20 10:59

c1++;
++c2;
c3++;

cout << c1 << " " << c2 << " " << c3 << endl;
// should output 12:01 03:21 11:00

c1 = ++c2;
cout << c1 << " " << c2 << endl;
// should output 03:22 03:22

c1 = c3++;
cout << c1 << " " << c3 << endl;
// should output 11:00 11:01

Clock c4(0,0);
cout << c1-- << " " << c4-- << endl;
// should output 11:00 00:00
cout << c1 << " " << c4 << endl;
// should output 10:59 23:59
cout << --c1 << " " << --c4 << endl;
// should output 10:58 23:58

return 0;
}
```

4.7. Dani (preklapanje operatora za enumeracije).

Ilustrovati preklapanje operatora za nabrajanja.

Rešenje:

```
// Nabrajanja i preklapanje operatora.

#include <iostream>
using namespace std;

enum Dan { PO, UT, SR, CE, PE, SU, NE};

inline Dan operator+ (Dan d, int k)
  { k = (int(d) + k) % 7; if (k < 0) k += 7; return Dan (k); }
inline Dan operator- (Dan d, int k) { return d + -k; }
inline Dan& operator+= (Dan& d, int k) { return d = d + k; }
inline Dan& operator-= (Dan& d, int k) { return d = d - k; }

inline Dan& operator++ (Dan& d) { return d = Dan (d<NE ? int(d)+1 : PO); }
inline Dan& operator-- (Dan& d) { return d = Dan (d>PO ? int(d)-1 : NE); }
inline Dan operator++ (Dan& d, int) { Dan e(d); ++d; return e; }
inline Dan operator-- (Dan& d, int) { Dan e(d); --d; return e; }

ostream& operator<< (ostream& dat, Dan dan) {
  char* dani[] = {"pon", "uto", "sre", "cet", "pet", "sub", "NED"};
  return dat << dani[dan];
}

int main () {
  for (Dan d=PO; d<NE; ++d)
    cout << d << ' ' << d+2 << ' ' << d-2 << endl;
}
```

5. Nasleđivanje klasa i polimorfizam

5.1. Osobe, đaci, zaposleni.

Osoba ima ime, datum rođenja i adresu stanovanja. Vrednosti ovih atributa mogu da se pročitaju sa standardnog ulaza i ispišu na standardni izlaz. Podrazumevano, pre čitanja, ove vrednosti ne postoje. *Đak* je osoba koja dodatno ima podatke o nazivu škole koju pohađa i razred u koji ide. Vrednosti atributa mu se čitaju sa standardnog ulaza, tako što se prvo učitaju podaci za osobu, a zatim i dodatni podaci o *Đaku*. Ovi podaci se na isti način i pišu se na standardni izlaz. *Zaposleni* je osoba koja ima dodatne podatke o preduzeću u kom radi i naziv odeljenja. Vrednosti atributa mu se čitaju sa standardnog ulaza, tako što se prvo učitaju osnovni podaci za osobu, a zatim i dodatni podaci o zaposlenju. Ovi podaci se na isti način i ispisuju na standardni izlaz. Napisati na programskoj jeziku C++ klase za opisane koncepte.

Napisati glavni program koji napravi nekoliko objekata ovih klasa, učita im vrednosti atributa, a zatim ih sve ispiše na standardnom izlazu.

Rešenje:

```
// osobe.h - Klase osoba, djaka i zaposlenih.
class Osoba {
  char ime[31], datum[11], adresa[31];
public:
  Osoba () { ime[0] = datum[0] = adresa[0] = 0; }
  virtual void citaj ();
  virtual void pisi  () const;
};

class Djak: public Osoba {
  char skola[31], razred[7];
public:
Djak (): Osoba() { skola[0] = razred[0] = 0; }
  void citaj ();
  void pisi  () const;
};

class Zaposlen: public Osoba {
  char firma[31], odeljenje[31];
public:
  Zaposlen (): Osoba() { firma[0] = odeljenje[0] = 0; }
  void citaj ();
  void pisi  () const;
};


// osobe.cpp - Metode klasa osoba, djaka i zaposlenih.

#include "osobe.h"
#include <iostream>
using namespace std;

void Osoba::citaj () {
  cout << "Ime i prezime?     "; cin >> ws;
                                  cin.getline (ime,    31);
  cout << "Datum rodjenja?    "; cin.getline (datum,  11);
  cout << "Adresa stanovanja? "; cin.getline (adresa, 31);
}
```

```
void Osoba::pisi  () const {
  cout << "Ime i prezime:     " << ime       << endl;
  cout << "Datum rodjenja:    " << datum     << endl;
  cout << "Adresa stanovanja: " << adresa    << endl;
}

void Djak::citaj () {
  Osoba::citaj ();
  cout << "Naziv skole?       "; cin.getline (skola, 31);
  cout << "Razred?            "; cin.getline (razred, 7);
}

void Djak::pisi  () const {
  Osoba::pisi ();
  cout << "Naziv skole:       " << skola     << endl;
  cout << "Razred:            " << razred    << endl;
}

void Zaposlen::citaj () {
  Osoba::citaj ();
  cout << "Naziv firme?       "; cin.getline (firma,     31);
  cout << "Naziv odeljenja?   "; cin.getline (odeljenje, 31);
}

void Zaposlen::pisi () const {
  Osoba::pisi ();
  cout << "Naziv firme:       " << firma     << endl;
  cout << "Naziv odeljenja:   " << odeljenje << endl;
}


// osobet.C - Ispitivanje klasa osoba, djaka i zaposlenih.
#include "osobe.h"
#include <iostream>
using namespace std;

int main () {
  Osoba* ljudi[20]; int n=0;

  cout << "Citanje podataka o ljudima\n";
  while (true) {
    cout << "\nIzbor (O=osoba, D=djak, Z=zaposlen, K=kraj)? ";
    char izbor; cin >> izbor;
  if (izbor=='K' || izbor=='k') break;
    ljudi[n] = 0;
    switch (izbor) {
      case 'O': case 'o': ljudi[n] = new Osoba;    break;
      case 'D': case 'd': ljudi[n] = new Djak;     break;
      case 'Z': case 'z': ljudi[n] = new Zaposlen; break;
    }
    if (ljudi[n]) ljudi[n++]->citaj();
  }

  cout << "\nPrikaz procitanih podataka\n";
  for (int i=0; i<n; i++) { cout << endl; ljudi[i]->pisi (); }

  return 0;
}
```

5.2. Apstraktna, putnička i teretna vozila (prelazak mosta).

Vozilo ima sopstvenu težinu, može da mu se pročita vrsta, da se odredi ukupna težina (podrazumenano jednaka sopstvenoj težini) i da se ispiše u izlazni tok podataka, tako što se ispiše oznaka vrste, a u zagradama navede sopstvena težina. Stvara se zadavanjem sopstvene težine. *Putničko vozilo* je vozilo koje prevozi putnike i ima podatak o broju putnika i težina prosečnog putnika. Stvara se zadavanjem sopstvene težine, broja putnika i njihove prosečne težine. Vrsta mu je označena sa 'P', ukupna težina se dobija sabiranjem sopstvene težine i težine putnika, a ispisuje se isto kao i Vozilo, s tim što se u zagradama navodi još i broj putnika i srednja težina putnika. *Teretno vozilo* je Vozilo koje prevozi teret zadate težine. Stvara se zadavanjem sopstvene težine i težine tereta. Ukupna težina mu se dobija sabiranjem sopstvene i težine tereta. Oznaka vrste je 'T'. Ispisuje se isto kao i Vozilo, s tim što se u zagradama, pored sopstvene težine, navodi I težina tereta.

Napisati na programskom jeziku C++ klase za opisane koncepte i glavni program koji testira njihove funkcionalnosti.

Rešenje:

```
// vozilo.h - Apstraktna klasa vozila.
#ifndef _vozilo_h_
#define _vozilo_h_

#include <iostream>
using namespace std;

class Vozilo
{
  double sopTez;                                    // Sopstvena tezina.

public:
  Vozilo (double st) { sopTez = st; }               // Konstruktor.
  virtual char vrsta () const =0;                   // Vrsta vozila.
  virtual double tezina () const { return sopTez; }  // Ukupna tezina.

  static Vozilo* napraviVozilo(int vrsta);         // Pravljenje vozila zadate vrste.

protected:
  virtual void pisi (ostream& it) const             // Pisanje.
    { it << vrsta() << '(' << sopTez << ','; }

  friend ostream& operator<< (ostream& it, const Vozilo& v) // vazi princip supstitucije.
    { v.pisi (it); return it; }
};

#endif
```

```
// pvozilo.h - Klasa putnickih vozila.
#ifndef _pvozilo_h_
#define _pvozilo_h_

#include "vozilo.h"

class PVozilo: public Vozilo
{
double srTez;                           // Srednja tezina putnika.
int brPut;                              // Broj putnika.

public:
PVozilo (double st, double srt, int bp)  // Konstruktor.
: Vozilo (st)
{ srTez = srt; brPut = bp; }

char vrsta () const { return 'P'; }                  // Vrsta vozila.

double tezina () const                               // Ukupna tezina.
{ return Vozilo::tezina () + srTez * brPut; }
private:
void pisi (ostream& it) const                        // Pisanje.
{ Vozilo::pisi (it); it << srTez << ',' << brPut << ')'; }
};

#endi

// tvozilo.h - Klasa teretnih vozila.
#ifndef _tvozilo_h_
#define _tvozilo_h_

#include "vozilo.h"

class TVozilo: public Vozilo
{
double teret;                                        // Tezina tereta.

public:
TVozilo (double st, double t)               // Konstruktor.
: Vozilo (st)
{ teret = t; }

char vrsta () const { return 'T'; }                  // Vrsta vozila.

double tezina () const                               // Ukupna tezina.
{ return Vozilo::tezina () + teret; }
private:
void pisi (ostream& it) const                        // Pisanje.
{ Vozilo::pisi (it); it << teret << ')'; }
};

#endif
```

```
// vozilat.cpp - Ispitivanje klasa vozila.
#include "tvozilo.h"
#include "pvozilo.h"
#include <iostream>
using namespace std;

Vozilo* Vozilo::napraviVozilo(int vrsta)
{
switch (vrsta)
{
case 't': case 'T':
   cout << "Sopstvena tezina?     "; double sTez;  cin >> sTez;
   cout << "Tezina tereta?        "; double ter;   cin >> ter;
   return new TVozilo (sTez, ter);
   break;
case 'p': case 'P':
   cout << "Sopstvena tezina?     ";               cin >> sTez;
   cout << "Sr. tezina putnika?   "; double srTez; cin >> srTez;
   cout << "Broj putnika?         "; int    brPut; cin >> brPut;
   return new PVozilo (sTez, srTez, brPut);
   break;
default:
   cout << "*** Nepoznata vrsta vozila!\n";
   return nullptr;
}
}

int main ()
{
Vozilo* vozila[100];
int n = 0;

while (true)
{
   cout << "\nVrsta vozila (T,P,*)? ";
   char vrsta;
   cin >> vrsta;

   if (vrsta == '*') break;

   Vozilo* v = Vozilo::napraviVozilo(vrsta);
   if (v != nullptr)
      vozila[n++] = v;
}
cout << "\nNosivost mosta?       ";
double nosivost;
cin >> nosivost;
cout << "\nMogu da predju most:\n";
for (int i=0; i<n; i++)
   if (vozila[i]->tezina() <= nosivost)
      cout << *vozila[i] << " - " << vozila[i]->tezina() << endl;

// Dealokacija memorije.
for (int i=0; i<n; i++)
   delete vozila[i];

return 0;
}
```

5.3. Predmeti sa specifičnom težinom, sfera, kvadar.

Predmet ima definisanu specifičnu težinu. Stvara se zadavanjem specifične težine (podrazumevano 1), može da mu se izračuna zapremina i težina, može da se učita iz ulaznog toka i ispiše u izlazni tok. *Kvadar* je predmet sa zadatom dužinom, širinom i visinom. Pravi se zadavanjem dimenzija (podrazumevano 1) i specifične težine. *Sfera* je predmet koji ima zadat poluprečnik. Stvara se zadavanjem dužine poluprečnika (podrazumevano 1) i specifične težine.

Napisati program na programskom jeziku C++ koji prvo učita nekoliko predmeta, izračuna njihovu srednju težinu, a zatim ispiše na standarni izlaz sve one čija je težina veća od prosečne. Na kraju, program treba da oslobodi sve zauzete resurse.

Rešenje:

```
// predmet1.h - Apstraktna klasa predmeta.

#ifndef _predmet1_h_
#define _predmet1_h_

#include <iostream>
using namespace std;

class Predmet
{
double sigma;        // Specificna tezina.

public:
Predmet (double ss=1) { sigma = ss; }                  // Konstruktor.
virtual double V () const =0;                          // Zapremina.
double q () const { return V () * sigma; }             // Tezina.

protected:
virtual void citaj (istream &t) { t >> sigma; }        // Citanje.
virtual void pisi (ostream &t) const { t << sigma; }   // Pisanje.

// Uopsteno citanje i pisanje (preklapanjem operatora).
friend istream& operator>> (istream& t, Predmet& p);
friend ostream& operator<< (ostream& t, const Predmet& p);

public:
static Predmet* napraviPredmet(char vrsta);
};


inline istream& operator>> (istream& t, Predmet &p)    // Uopsteno citanje
{
p.citaj (t);
return t;
}

inline ostream& operator<< (ostream& t, const Predmet &p) // Uopsteno pisanje.
{
p.pisi (t);
return t;
}

#endif
```

```
   // sfera1.h - Klasa sfera.
   #ifndef _sfera1_h_
   #define _sfera1_h_

   #include "predmet1.h"

   class Sfera: public Predmet
   {
   double r;  // Poluprecnik.

   public:
   Sfera (double ss=1, double rr=1)                        // Konstruktor.
   : Predmet (ss)
   { r = rr; }

   double V() const { return 4./3 * r*r*r * 3.14159; }     // Zapremina.

   private:
   void citaj (istream& t);                                // Citanje.
   void pisi  (ostream& t) const;                          // Pisanje.
   };

   #endif


   // kvadar3.h - Klasa kvadara.
   #ifndef _kvadar3_h_
   #define _kvadar3_h_

   #include "predmet1.h"

   class Kvadar: public Predmet
   {
   double a, b, c; // Dimenzije.

   public:
   Kvadar (double ss=1, double aa=1, double bb=1, double cc=1) // Konstr.
   : Predmet (ss)
   { a = aa; b = bb; c = cc; }

   double V () const { return a * b * c; }                 // Zapremina.

   private:
   void citaj (istream& t);                                 // Citanje.
   void pisi  (ostream& t) const;                          // Pisanje.
   };

#endif
```

```cpp
/*
 * predmet1.cpp - Definicije metoda iz klasa predmeta.
 *
 *  Created on: 10.12.2014.
 *      Author: Nemanja Kojic
 */

#include "sfera1.h"

void Sfera::citaj (istream& t)
{
Predmet::citaj (t);
t >> r;
}

void Sfera::pisi  (ostream& t) const
{
t << "sfera [";
Predmet::pisi (t);
t << ',' << r << ']';
}


#include "kvadar3.h"

void Kvadar::citaj (istream& t)
{
Predmet::citaj (t);
t >> a >> b >> c;
}

void Kvadar::pisi  (ostream& t) const
{
t << "kvadar[";
Predmet::pisi (t);
t << ',' << a << ',' << b << ',' << c << ']';
}


#include "predmet1.h"

Predmet* Predmet::napraviPredmet(char vrsta)
{
switch (vrsta)
{
case 's': case 'S':
   return new Sfera;
case 'k': case 'K':
   return new Kvadar;
default:
   return nullptr;
}
}
```

```cpp
// predmet1t.C - Ispitivanje klasa predmeta.
#include "predmet1.h"
#include "kvadar3.h"
#include "sfera1.h"
#include <iomanip>
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
vector<Predmet*> predmeti;
double procecnaTezina = 0;

bool dalje = true;
while (dalje)
{
   char tip;
   cin >> tip;
   if (tip == '.') break;

   Predmet* p = Predmet::napraviPredmet(tip);
   if (p == nullptr)
      dalje = false;

   if (dalje && p)
   {
      cin >> *p;
      predmeti.push_back(p);
      cout << *p << " (q=" << p->q() << ")\n";
      procecnaTezina += p->q();
   }
}

if (predmeti.size() > 0)
   procecnaTezina /= predmeti.size();
cout << "\ngsr= " << procecnaTezina << "\n\n";

// Prikazivanje sadrzaja liste na glavnom izlazu.
// predmetIt sadrzi adresu podatka tipa Predmet*, zato je *predmetIt po tipu Predmet*
// i zato je za pristup primerku klase Predmet potrebno da se ponovi dereferenciranje.
for (vector<Predmet*>::iterator predmetIt = predmeti.begin();
        predmetIt != predmeti.end();
        ++predmetIt) // Prefiksno inkrementiranje iteratora!
{
   Predmet *p = *predmetIt;
   if (p->q() > procecnaTezina)
      cout << *p << " (q=" << p->q() << ")\n";
}

// Unistavanje prve liste. Prvo treba unistiti objekte na koje pokazuju pokazivaci iz liste.
for (auto predmetIt = predmeti.begin(); predmetIt != predmeti.end(); ++predmetIt)
   {  delete (*predmetIt);  }
// Potom treba izbaciti same pokazivace iz liste.
// Ovaj korak nije neophodan jer std::list u svom destruktoru radi ovaj posao.
predmeti.clear();
return 0;
}
```

5.4. Figure u ravni.

Tačka u dvodimenzionalnom prostoru opisana je realnim koordinatama **x** i **y**, koje mogu i da se pročitaju. Stvara se zadavanjem vrednosti koordinata (podrazumevano 0). Tačka se može učitati sa standardnog ulaza i ispisati na standardni izlaz. Apstraktna *figura* u ravni ima zadatu tačku težišta. Stvara se zadavanjem tačke težišta (podrazumevano u koordinatnom početku), koje može naknadno da se promeni. Moguće je figuru pomeriti sa zadatim vrednostima pomeraja, izračunati joj obim i površinu, učitati je sa standardnog ulaza i ispisati na standardni izlaz (preklapanjem operatora). Figura može da se dinamički kopira. *Krug* je figura koja ima dodatno definisan poluprečnik. Stvara se zadavanjem dužine poluprečnika (podrazumevano 1) i položaja centra (težišta, podrazumevano koordinatni početak). *Kvadrat* je figura sa definisanom veličinom stranice. Stvara se zadavanjem dužine stranice (podrazumevano 1) i položajem težišta (podrazumevano koordinatni početak). *Trougao* je figura opisana dužinama stanica. Stvara se zadavanjem dužina stranica (podrazumevano 1) i položajem težišta (podrazumevano koordinatni početak). Omogućiti stvaranje sledećih vrsta trouglova definisanjem odgovarajućih konstruktora: jednakostranični, jednakokraki, opšti.

Napisati program na programskom jeziku C++ koji prvo učita nekoliko figura, ispiše ih na standarni izlaz, zatim svaku figuru pomeri za zadati pomeraj i ispiše novo stanje figura u ravni. Na kraju, program treba da oslobodi sve zauzete resurse.

Rešenje:

```
// Definicija klase tacaka (Tacka).
#ifndef _tacka2_h_
#define _tacka2_h_

typedef double Real;           // Tip za realne vrednosti.

#include <iostream>
using namespace std;

class Tacka {
  Real x, y;                                              // Koordinate.
public:
  Tacka (Real xx=0, Real yy=0) { x = xx; y = yy; }        // Konstruktor.
  Real aps () const { return x; }                         // Apscisa.
  Real ord () const { return y; }                         // Ordinata.
  friend istream& operator>> (istream& dd, Tacka& tt)     // Citanje.
    { return dd >> tt.x >> tt.y; }
  friend ostream& operator<< (ostream& dd, const Tacka& tt) // Pisanje.
    { return dd << '(' << tt.x << ',' << tt.y << ')'; }
} ;

const Tacka ORG;               // Koordinatni pocetak.

#endif
```

```
// Definicija klase geometrijskih figura (Figura).
#ifndef _figura1_h_
#define _figura1_h_

#include "tacka2.h"

class Figura {
  Tacka T;                                  // Teziste figure.
public:
  Figura (const Tacka& tt=ORG): T(tt)  {}   // Konstruktor.
  virtual ~Figura () {}                     // Destruktor.
  virtual Figura* kopija () const =0;       // Stvaranje kopije.
  Figura& postavi (Real xx, Real yy)        // Postavljanje figure.
    { T = Tacka (xx, yy); return *this; }
  Figura& pomeri (Real dx, Real dy)         // Pomeranje figure.
    { T = Tacka ( T.aps() + dx, T.ord() + dy); return *this; }
  virtual Real O () const =0 ;              // Obim.
  virtual Real P () const =0 ;              // Povrsina.
protected:
  virtual void citaj (istream& dd)       { dd >> T; }          // Citanje.
  virtual void pisi  (ostream& dd) const { dd << "T=" << T; }  // Pisanje.
  friend istream& operator>> (istream& dd, Figura& ff)  // Uopsteno citanje
    { ff.citaj (dd); return dd; }                               //    i
  friend ostream& operator<< (ostream& dd, const Figura& ff)   // pisanje.
    { ff.pisi  (dd); return dd; }
} ;
#endif

// Definicija klase krugova (Krug).
#ifndef _krug2_h_
#define _krug2_h_

#include "figura1.h"

namespace { const Real PI = 3.14159265359; }

class Krug : public Figura {
  Real r;                                   // Poluprecnik.
public:
  Krug (Real rr=1, const Tacka& tt=ORG)     // Konstruktor.
    : Figura (tt) { r = rr; }
  Krug* kopija () const                     // Stvaranje kopije.
    { return new Krug (*this); }
  Real O () const { return 2 * r * PI; }    // Obim.
  Real P () const { return r * r * PI; }    // Povrsina.
private:
  void citaj (istream& dd) ;                // Citanje.
  void pisi  (ostream& dd) const ;          // Pisanje.
} ;
#endif

// Definicije metoda uz klasu Krug.
#include "krug2.h"
void Krug::citaj (istream& dd)                // Citanje.
  { Figura::citaj (dd); dd >> r ; }

void Krug::pisi  (ostream& dd) const {        // Pisanje.
  dd << "krug    ["; Figura::pisi (dd);
  dd << ", r=" << r << ", O=" << O() << ", P=" << P() << ']';
}
```

```
// Definicija klase kvadrata (Kvadrat).
#ifndef _kvadrat_h_
#define _kvadrat_h_

#include "figura1.h"

class Kvadrat : public Figura {
  Real a;                                   // Osnovica.
public:
  Kvadrat (Real aa=1, const Tacka& tt=ORG) // Konstruktor.
    : Figura(tt) { a = aa; }
  Kvadrat* kopija () const                  // Stvaranje kopije.
    { return new Kvadrat (*this); }
  Real O () const { return 4 * a; }         // Obim.
  Real P () const { return a * a; }         // Povrsina.
private:
  void citaj (istream& dd) ;                // Citanje.
  void pisi  (ostream& dd) const ;          // Pisanje.
} ;

#endif

// Definicije metoda uz klasu Kvadrat.
#include "kvadrat.h"

void Kvadrat::citaj (istream& dd)           // Citanje.
  { Figura::citaj (dd); dd >> a ; }

void Kvadrat::pisi  (ostream& dd) const {   // Pisanje.
  dd << "kvadrat [";
  Figura::pisi (dd);
  dd << ", a=" << a << ", O=" << O() << ", P=" << P() << ']';
}

// Definicija klase trouglova (Trougao).
#ifndef _trougao1_h_
#define _trougao1_h_

#include "figura1.h"

class Trougao : public Figura {
  Real a, b, c;                             // Strane.
public:                                     // Konstruktori:
  Trougao (Real aa=1, const Tacka& tt=ORG)          // jednakostranicni
    : Figura (tt) { a = b = c = aa; }
  Trougao (Real aa, Real bb, const Tacka& tt=ORG)   // jednakokraki
    : Figura (tt) { a = aa; b = c = bb; }
  Trougao (Real aa, Real bb, Real cc, const Tacka& tt=ORG) // opsti
    : Figura (tt) { a = aa; b = bb; c = cc; }
  Trougao* kopija () const                  // Stvaranje kopije.
    { return new Trougao (*this); }
  Real O () const { return a + b + c; }     // Obim.
  Real P () const ;                         // Povrsina.
private:
  void citaj (istream& dd) ;                // Citanje.
  void pisi  (ostream& dd) const ;          // Pisanje.
} ;

#endif
```

```
// Definicije metoda uz klasu Trougao.
#include "trougao1.h"
#include <cmath>
using namespace std;

Real Trougao::P () const {                  // Povrsina.
  Real s = (a + b + c) / 2;
  return sqrt (s * (s-a) * (s-b) * (s-c));
}

void Trougao::citaj (istream& dd)           // Citanje.
  { Figura::citaj (dd); dd >> a >> b >> c; }

void Trougao::pisi  (ostream& dd) const {   // Pisanje.
  dd << "trougao [";
  Figura::pisi (dd);
  dd << ", a=" << a << ", b=" << b   << ", c=" << c
     << ", O=" << O() << ", P=" << P() << ']';
}

// Program za ispitivanje klasa za geometrijske figure.
// Ujedno pokazuje rad sa klasom std::list.
#include <iostream>
#include <iomanip>
#include <list>
using namespace std;

#include "krug2.h"
#include "kvadrat.h"
#include "trougao1.h"

int main ()
{
list<Figura*> figure;
// Stvaranje liste figura citajuci preko glavnog ulaza.
for (bool dalje=true; dalje; )
{
   Figura* pf;
   char vrsta; cin >> vrsta;
   switch (vrsta)
   {
   case 'o': pf = new Krug;    break;
   case 'k': pf = new Kvadrat; break;
   case 't': pf = new Trougao; break;
   default: dalje = false;
   }
   if (dalje)
   {
     cin >> *pf;
     figure.push_back(pf);
   }
}

// Prikazivanje sadrzaja liste na glavnom izlazu.
// Prvo adresa podatka, a onda i sam podatak.\
// figIt sadrzi adresu podatka tipa Figura*, zato je *figIt po tipu Figura*
// i zato je za pristup primerku klase Figura potrebno da se ponovi dereferenciranje.
// Dakle, *figIt je adresa figure, a **figIt je sama figura.
for (list<Figura*>::iterator figIt = figure.begin(); figIt != figure.end(); figIt++)
   cout << left << setw(10) << *figIt << *(*figIt) << endl;
```

```
// Citanje pomeraja za pomeranje figura.
Real dx, dy; cin >> dx >> dy;
cout << "\ndx, dy= " << dx << ", " << dy << "\n\n";

// Stvaranje kopije liste uz pomeranje figura.
// Realizovano kopiranjem jednog po jednog elementa, sa pomocnim pokazivacem
list<Figura*> pomereneFigure;
for (list<Figura*>::iterator figIt = figure.begin(); figIt != figure.end(); figIt++)
{
  Figura *pf = (*figIt);
  Figura *pfNova = pf->kopija(); // isto bi bilo da stoji = (*figIt)->kopija()
  pfNova->pomeri(dx, dy);
  pomereneFigure.push_back(pfNova);
}
// OPREZ: koriscenje konstruktora kopije ili operatora dodele za std::list,
// na primer:
// list<Figura*> pomereneFigure(figure); ili josJednaListaFigura = figure;
// bi iskopiralo sadrzane pokazivace, ali NE I OBJEKTE NA KOJE ONI POKAZUJU!!!
// Tako bi obe liste pokazivale na iste objekte.
// Ovo ne mora biti greska, ali zahteva da programer vodi racuna o tome
// da objekte oslobadja samo jednom, a ne prilikom svakog unistavanja liste.
// Za liste koje sadrze primerke klasa, a ne pokazivace,
// ovo razmatranje nije potrebno, posto ce kod njih lista u svom destruktoru
  // sama unistavati sadrzane primerke klasa.

// Unistavanje prve liste.
// Prvo treba unistiti objekte na koje pokazuju pokazivaci iz liste.
for (list<Figura*>::iterator figIt = figure.begin(); figIt != figure.end(); figIt++)
{
  delete (*figIt); // isto bi bilo da stoji Figura *pf = *figIt; delete pf;
}
// Potom treba izbaciti same pokazivace iz liste.
// Ovaj korak nije neophodan jer std::list u svom destruktoru radi ovaj posao.
figure.clear();

// Prikazivanje sadrzaja kopirane liste na glavnom izlazu.
for (list<Figura*>::iterator pomFigIt = pomereneFigure.begin();
     pomFigIt != pomereneFigure.end(); pomFigIt++)
  cout << **pomFigIt << endl;

// Unistavanje liste pomerenih figura.
// Prvo treba unistiti objekte na koje pokazuju pokazivaci iz liste.
for (list<Figura*>::iterator pomFigIt = pomereneFigure.begin();
     pomFigIt != pomereneFigure.end(); pomFigIt++)
{
  delete (*pomFigIt); // isto bi bilo da stoji Figura *pf = *figIt; delete pf;
}
}
```

6. Standardna biblioteka

6.1. Standardna kolekcija std::vector

Sledeći program napisan na programskom jeziku C++ ilustruje rad sa klasom **`vector`** iz standardne biblioteke.

Rešenje:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Tip elementa vektora može biti bilo šta,
// što ima definisan podrazumevani konstruktor, konstruktor kopije,
// destruktor i operatore =, < i ==.
// Za proste tipove (brojeve, znakove, pokazivače) to je već zadovoljeno.
typedef int VectorElem;

// Ova klasa će biti upotrebljena za definisanje raznih operatora ()
// preko kojih će biti obavljani potrebni proračuni i prebrojavanja.
class Helper {
public:
   // Konstruktor je samo inicijalizator.
   Helper() : evenCount(0), oddCount(0) { }

   // Brojači parnih i neparnih elemenata vektora. Nema potrebe da budu privatni.
   unsigned evenCount;
   unsigned oddCount;

   // operator() koji prima elemente vektora i onda ih prebroji.
   // Bitno je da operator prima podatke tipa elementa vektora
   // Povratna vrednost operatorske funkcije nigde neće biti upotrebljena.
   // Ako je potrebno da element vektora bude promenjen, treba ga preneti po referenci.
   unsigned operator() (VectorElem pv_vectorElem)
   {
      // NAPOMENA: klasa VectorElem mora imati adekvatno definisan operator % da bi ovaj
      // primer mogao biti preveden.
      return pv_vectorElem % 2 ? ++oddCount : ++evenCount;
   }
};

// Slično kao klasa, za for_each
// može biti upotrebljena i funkcija koja prihvata element vektora.
// Ovo treba raditi ako zbirni efekat funkcije nije od značaja,
// te nema potrebe pamtiti bilo šta u namenskom funkcijskom objektu.
// Funkcija može biti bilo kog tipa.
// Funkcija može raditi bilo šta.
void writeElem(VectorElem pv_vectorElem) {
   // NAPOMENA: za klasu elementa vektora mora biti deklarisan operator << da bi ovaj
   // primer mogao biti preveden.
   cout << pv_vectorElem << endl;
}

// Ako je potrebno da element vektora bude promenjen,
// element treba preneti po referenci.
int makeSquare(VectorElem& pr_vectorElem) {
   return pr_vectorElem *= pr_vectorElem;
}

int main() {
   vector<VectorElem> brojevi;
```

```
    // Dodavanje elemenata u vektor.
    brojevi.push_back(1);
    brojevi.push_back(22);
    // Pre upotrebe operatora [] obavezno je da vektor ima element na traženom mestu
    brojevi.resize(3);
    brojevi[2] = 33;
    // brojevi.push_back(33); // ovo je bolje za dodavanje na kraj vektora, pošto
    // automatski povećava vektor
    brojevi[0] = 11; // ovo je izmena postojećeg elementa, za vrednost indeksa dolazi u
                     // obzir bilo koja vrednost između 0 ili size()-1

    // Ispis elemenata.
    cout << "Vektor brojeva" << endl;
    for_each(brojevi.begin(), brojevi.end(), writeElem);

    // Kvadriranje elemenata i ispis elemenata.
    for_each(brojevi.begin(), brojevi.end(), makeSquare);
    cout << "Vektor kvadrata" << endl;
    for_each(brojevi.begin(), brojevi.end(), writeElem);

// Treći argument for_each može biti i pomoćni objekat
// koji mora imati operator () definisan sa jednim argumentom,
// tipa podatka sadržanog u zbirci na koju se primenjuje for_each.
// for_each poziva operator () tog pomoćnog objekta za svaki od elemenata u zbirci.
// Vrednost funkcije for_each je kopija tog pomoćnog objekta.
// Postoje dva načina za upotrebu pomoćnog objekta.
// U obe varijante, obavezno je dodeliti rezultat funkcije for_each nekom objektu
// pomoćne klase sa opisanim operatorom () (u ovom primeru, klasa Helper), zato što
// for_each treći argument uzima po vrednosti i rezultat vraća po vrednosti,
// tako da pomoćni objekat na mestu trećeg argumenta neće biti izmenjen,
// niti će bilo kakve promene nad kopijom tog objekta biti zapamćene.
// Ako je potrebno prvo formirati pomoćni objekat, koristiti 1. pristup.

// 1. Prvo napraviti primerak klase, pa taj primerak zadati kao argument for_each.
    // Za svaki element zbirke biće pozvan elementCounter(*it), gde je it iterator koji
// se kreće od brojevi.begin() do brojevi.end(), ne uključujući brojevi.end().
    Helper elementCounter;
// ovde po potrebi dodati pripremu elementCounter za poziv for_each().
// ...
    elementCounter = for_each(brojevi.begin(), brojevi.end(), elementCounter);
    cout << "U vektoru ima " << elementCounter.evenCount << " parnih elemenata" << endl;
    cout << "U vektoru ima " << elementCounter.oddCount << " neparnih elemenata" << endl;

    // 2. for_each pozvati sa privremenim objektom klase Helper kao argumentom.
    // Za svaki element zbirke biće pozvan operator() od kopije privremenog objekta sa
    // argumentom (*it).
    Helper anotherCounter = for_each(brojevi.begin(), brojevi.end(), Helper::Helper());
    cout << "U vektoru ima " << anotherCounter.evenCount << " parnih elemenata" << endl;
    cout << "U vektoru ima " << anotherCounter.oddCount << " neparnih elemenata" << endl;

    return 0;
}
```

6.2. Standardna kolekcija std::list.

Sledeći program napisan na programskom jeziku C++ ilustruje rad sa klasom **`list`** iz standardne biblioteke.

Rešenje:

```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

// Tip elementa liste može biti bilo šta,
// što ima definisan podrazumevani konstruktor, konstruktor kopije,
// destruktor i operatore =, < i ==.
// Za proste tipove (brojeve, znakove, pokazivače) to je već zadovoljeno.
typedef int ListElem;

// Ova klasa će biti upotrebljena za definisanje raznih operatora ()
// preko kojih će biti obavljani potrebni proračuni i prebrojavanja.
class Helper
{
public:
// Konstruktor je samo inicijalizator.
Helper() : evenCount(0), oddCount(0)
{
}

// Brojači parnih i neparnih elemenata liste. Nema potrebe da budu privatni.
unsigned evenCount;
unsigned oddCount;

// operator() koji prima elemente liste i onda ih prebroji.
// Bitno je da operator prima podatke tipa elementa liste
// Povratna vrednost operatorske funkcije nigde neće biti upotrebljena.
// Ako je potrebno da element liste bude promenjen, treba ga preneti po referenci.
unsigned operator() (ListElem pv_listElem)
{
     // NAPOMENA: klasa ListElem mora imati adekvatno definisan operator % da bi ovaj
     // primer mogao biti preveden.
   return pv_listElem % 2 ? ++oddCount : ++evenCount;
}
};

// Slično kao klasa, za for_each
// može biti upotrebljena i funkcija koja prihvata element liste.
// Ovo treba raditi ako zbirni efekat funkcije nije od značaja,
// te nema potrebe pamtiti bilo šta u namenskom funkcijskom objektu.
// Funkcija može biti bilo kog tipa.
// Funkcija može raditi bilo šta.
void writeElem(ListElem pv_listElem)
{
// NAPOMENA: za klasu elementa liste mora biti deklarisan operator << da bi ovaj primer
mogao biti preveden.
cout << pv_listElem << endl;
}

// Ako je potrebno da element liste bude promenjen, element treba preneti po referenci.
int makeSquare(ListElem& pr_listElem)
{
return pr_listElem *= pr_listElem;
}
```

```
int main()
{
list<ListElem> brojevi;

// Dodavanje elemenata u listu.
brojevi.push_back(22);
brojevi.push_front(11);
brojevi.push_back(33);


// Ispis elemenata.
cout << "Lista brojeva" << endl;
for_each(brojevi.begin(), brojevi.end(), writeElem);

// Kvadriranje elemenata i ispis elemenata.
for_each(brojevi.begin(), brojevi.end(), makeSquare);
cout << "Lista kvadrata" << endl;
for_each(brojevi.begin(), brojevi.end(), writeElem);

// Treći argument for_each može biti i pomoćni objekat
// koji mora imati operator () definisan sa jednim argumentom,
// tipa podatka sadržanog u zbirci na koju se primenjuje for_each.
// for_each poziva operator () tog pomoćnog objekta za svaki od elemenata u zbirci.
// Vrednost funkcije for_each je kopija tog pomoćnog objekta.
// Postoje dva načina za upotrebu pomoćnog objekta.
// U obe varijante, obavezno je dodeliti rezultat funkcije for_each
// nekom objektu pomoćne klase sa opisanim operatorom ()
// (u ovom primeru, klasa Helper),
// zato što for_each treći argument uzima po vrednosti,
// i rezultat vraća po vrednosti,
// tako da pomoćni objekat na mestu trećeg argumenta neće biti izmenjen,
// niti će bilo kakve promene nad kopijom tog objekta biti zapamćene.
// Ako je potrebno prvo formirati pomoćni objekat, koristiti 1. pristup.

// 1. Prvo napraviti primerak klase, pa taj primerak zadati kao argument for_each.
// Za svaki element zbirke biće pozvan elementCounter(*it),
// gde je it iterator koji se kreće od brojevi.begin() do brojevi.end(),
// ne uključujući brojevi.end().
Helper elementCounter;
// ovde po potrebi dodati pripremu elementCounter za poziv for_each().
// ...
elementCounter = for_each(brojevi.begin(), brojevi.end(), elementCounter);
cout << "U listi ima " << elementCounter.evenCount << " parnih elemenata" << endl;
cout << "U listi ima " << elementCounter.oddCount << " neparnih elemenata" << endl;

// 2. for_each pozvati sa privremenim objektom klase Helper kao argumentom.
// Za svaki element zbirke biće pozvan operator() od kopije privremenog objekta sa
argumentom (*it).
Helper anotherElementCounter = for_each(brojevi.begin(), brojevi.end(),   );
cout << "U listi ima " << anotherElementCounter.evenCount << " parnih elemenata" <<
endl;
cout << "U listi ima " << anotherElementCounter.oddCount << " neparnih elemenata" <<
endl;

return 0;
}
```

6.3. Zadatak

Sledeći program napisan na programskom jeziku C++ ilustruje rad sa klasom **`map`** iz standardne biblioteke.

Rešenje:

```
// Zaglavlje za std::map i std::multimap (potonja zbirka nije bitna za kurs OOP)
#include <map>
// Standardni ulaz i izlaz
#include <iostream>
// Stringovi iz jezika C (char *, završna nula)
#include <cstring>
// Stringovi iz STL (znatno lakši za upotrebu)
#include <string>
using namespace std;

// Radi jednostavnijeg pisanja, programer može definisati svoje tipove
// Kod std::map, prvi argument šablona je tip ključa (engl. key),
// drugi argument šablona je tip vrednosti podatka (engl. value)
typedef map<int, char*> TMap;

// NAPOMENA: Iteratori nisu neophodni za najosnovniju upotrebu klase std::map.
typedef TMap::iterator TMapIt;

int main()
{
TMap myMap, newMap;

// Operator [] dohvata podatak pridružen ključu.
// Ako podatka nema, biće stvoren par koji će sadrzati ključ i podatak
// dobijen pozivom podrazumevanog konstruktora klase podatka za klasne tipove,
// odnosno podatak sa vrednošću 0 za proste tipove.
myMap[1];
// U svakom slučaju, rezultat operatora je referenca na podatak koji odgovara datom ključu.
// Zato operator [] praćen operatorom = čiji je drugi operand tipa podatka
// može poslužiti za dodavanje i dodelu vrednosti podataka.
// Dodavanje u mapu preko operatora [].
myMap[1] = "Pera";
myMap[123] = "Mika";
myMap[-12] = "Joca";

// Ako je potrebno ispisati samo vrednosti podataka, može opet biti upotrebljen operator [].
// Za pristup određenom podatku, potrebno je samo navesti vrednost ključa.
cout << myMap[123] << endl << endl;

// Metoda insert će imati isti efekat kao i operator [] praćen operatorom dodele, ali je sintaksa neugodnija.
myMap.insert(pair<int, char*>(54321, "Zika"));

// Prolazak kroz mapu je isti kao prolazak kroz ostale STL zbirke.
// Podaci u mapi su uređeni na osnovu operatora < za klasu ključa.
// Polje first je ključ, polje second je vrednost.
for (TMapIt it = myMap.begin(); it != myMap.end(); it++)
{
  cout << it->first << " " << it->second << endl;
}
cout << endl;
```

```
  // Za ispis samo vrednosti u celoj mapi, pogodna je ovakva petlja.
  for (TMapIt it = myMap.begin(); it != myMap.end(); it++)
  {
    cout << myMap[it->first] << endl;
  }
  cout << endl;

  // Gornji ispis radi zato što su svi pokazivani stringovi ("Pera", ...)
  // još uvek u memoriji kad dođe do ispisa (u istom su dosegu).
  // Nema garancije da će ovaj programski kod raditi ispravno
  // ako mapa sa ovim podacima bude upotrebljena u drugom dosegu,
  // zato što mapa sadrži samo pokazivače, a ne i znakove stringova.
  // Jedno rešenje je da se u mapu umeću celi dinamički stringovi.
  char* newValue = new char[100];
  strcpy(newValue, "Dule");
  myMap[27] = newValue;

  // Za zbirku čiji podaci neće biti menjani, treba koristiti iterator za nepromenljive
podatke
  // U ovom zadatku to nije bitno.
  TMap::const_iterator it1 = myMap.begin(), it2 = myMap.end();
  for ( ; it1 != it2; it1++)
  {
    cout << it1->first << " " << it1->second << endl;
  }
  cout << endl;

  // OPREZ: Kod ovakvog pristupa, programer mora obezbediti
  // brisanje dinamičkih C stringova prilikom uništavanja mape.
  // OPREZ: Programer mora voditi računa da prilikom umetanja C stringova u mapu
  // umeće ili samo dinamičke C stringove (new char ...)
  // ili samo konstantne stringove ("Pera", ...),
  // da bi uništavanje mape bilo unificirano.
  // Imajući sve to u vidu, ovde je potrebno uništiti samo poslednji umetnuti podatak.
  delete myMap[27];
  // Ostali podaci će biti uništeni u destruktoru mape ili
  // nakon eksplicitnog poziva metode clear.

  // Znatno bolje rešenje je koristiti klasu std::string,
  // koja se sama stara o sopstvenoj propisnoj alokaciji i dealokaciji,
  // i opremljena je odgovarajućim konstruktorima konverzije,
  // uključujući i konverziju od običnog C stringa.
  map<int, string> myBetterMap;

  // Dodavanje u mapu preko operatora [] praćenog operatorom =.
  // STL string moze biti napravljen na razne načine.
  // Konstruktor konverzije, implicitno.
  myBetterMap[1] = "Pera";
  // Konstruktor konverzije, eksplicitno.
  myBetterMap[123] = string("Mika");
  // Prvo konverzija, pa onda dodavanje u mapu
  char *cStyleString = "Joca";
  string s = cStyleString;
  myBetterMap[-12] = s;

  // Metoda insert će imati isti krajnji efekat kao i operator []
  // praćen operatorom dodele, ali je sintaksa neugodnija.
  myBetterMap.insert(pair<int, string>(54321, string("Zika")));
```

```
// Prolazak kroz mapu je isti kao prolazak kroz ostale STL zbirke.
// Podaci u mapi su uređeni na osnovu operatora < za klasu ključa.
for (map<int, string>::iterator it = myBetterMap.begin();
     it != myBetterMap.end(); it++)
{
   cout << it->first << " " << it->second << endl;
}
cout << endl;

return 0;
}
```

6.4. Standardne kolekcije (kopiranje i sortiranje)

Sledeći program napisan na programskom jeziku C++ ilustruje pravljenje liste kopiranjem podataka iz vektora, kao i korišćenje postojećih algoritama za sortiranje.

Rešenje:

```
#include <iostream>
#include <list>
#include <vector>
#include <algorithm>
using namespace std;

// Tip elementa liste i vektora može biti bilo šta,
// što ima definisan podrazumevani konstruktor, konstruktor kopije,
// destruktor i operatore =, < i ==.
// Za proste tipove (brojeve, znakove, pokazivače) to je već zadovoljeno.
typedef int ListElem;
typedef ListElem VectorElem;

// Ova klasa će biti upotrebljena za definisanje raznih operatora ()
// preko kojih će biti obavljani potrebni proračuni i prebrojavanja.
class Helper
{
public:
// Konstruktor je samo inicijalizator.
Helper() : evenCount(0), oddCount(0)
{
}

// Brojači parnih i neparnih elemenata liste. Nema potrebe da budu privatni.
unsigned evenCount;
unsigned oddCount;

// operator() koji prima elemente liste i onda ih prebroji.
// Bitno je da operator prima podatke tipa elementa liste
// Povratna vrednost operatorske funkcije nigde neće biti upotrebljena.
// Ako je potrebno da element liste bude promenjen, treba ga preneti po referenci.
unsigned operator() (ListElem pv_listElem)
{
   // NAPOMENA: klasa ListElem mora imati adekvatno definisan operator %
   // da bi ovaj primer mogao biti preveden.
   return pv_listElem % 2 ? ++oddCount : ++evenCount;
}
};

// Slično kao klasa, za for_each
// može biti upotrebljena i funkcija koja prihvata element liste.
// Ovo treba raditi ako zbirni efekat funkcije nije od značaja,
// te nema potrebe pamtiti bilo šta u namenskom funkcijskom objektu.
// Funkcija može biti bilo kog tipa.
// Funkcija može raditi bilo šta.
void writeElem(ListElem pv_listElem)
{
// NAPOMENA: za klasu elementa liste mora biti deklarisan operator <<
// da bi ovaj primer mogao biti preveden.
cout << pv_listElem << endl;
}
```

```
int main()
{
list<ListElem> listaBrojeva;

// Dodavanje elemenata u listu.
listaBrojeva.push_back(22);
listaBrojeva.push_front(11);
listaBrojeva.push_back(33);
listaBrojeva.push_front(10000);
listaBrojeva.push_back(1);

// Ispis elemenata.
cout << "Lista" << endl;
for_each(listaBrojeva.begin(), listaBrojeva.end(), writeElem);

// Prebrojavanje elemenata po parnosti.
Helper elementCounter = for_each(listaBrojeva.begin(), listaBrojeva.end(),
Helper::Helper());
cout << "U listi ima " << elementCounter.evenCount << " parnih elemenata" << endl;
cout << "U listi ima " << elementCounter.oddCount << " neparnih elemenata" << endl;

// Lista može biti uređena svojom metodom sort.
listaBrojeva.sort();
for_each(listaBrojeva.begin(), listaBrojeva.end(), writeElem);

// Uređivanje može biti i drugačije, ako korisnik zada drugačiji kriterijum
// kao argument metode sort, u vidu funkcijskog objekta.
listaBrojeva.sort(greater<ListElem>());
for_each(listaBrojeva.begin(), listaBrojeva.end(), writeElem);

// Kopiranje jedne zbirke u drugu je jednostavno
// prvo treba napraviti npr. vektor odgovarajućeg kapaciteta
// i odgovarajućeg tipa. Mora postojati nedvosmislena konverzija od
// elemenata izvorišne do elemenata odredišne zbirke.
vector<VectorElem> vektorBrojeva(listaBrojeva.size());

// copy(prviIt, drugiIt, treciIt)
// radi sa intervalom [prviIt, drugiIt) i smešta počev od treciIt.
// Prvi i drugi iterator moraju biti od iste zbirke.
// Odredišna zbirka mora imati odgovarajući broj elemenata,
// pošto copy ne menja veličinu odredišne zbirke.
copy(listaBrojeva.begin(), listaBrojeva.end(), vektorBrojeva.begin());

cout << "Vektor" << endl;
for_each(vektorBrojeva.begin(), vektorBrojeva.end(), writeElem);

Helper anotherElementCounter = for_each(vektorBrojeva.begin(), vektorBrojeva.end(),
Helper::Helper());
cout << "U vektoru ima " << anotherElementCounter.evenCount << " parnih elemenata" <<
endl;
cout << "U vektoru ima " << anotherElementCounter.oddCount << " neparnih elemenata" <<
endl;

// Uređenje sa podrazumevanim kriterijumom koristi less<VectorElem>,
// što je podrazumevano operator < za VectorElem,
// što se u ovom primeru svodi na operator < za int.
sort (vektorBrojeva.begin(), vektorBrojeva.end());
for_each(vektorBrojeva.begin(), vektorBrojeva.end(), writeElem);
```

```
// Uređenje sa drugačijim kriterijumom u ovom primeru koristi greater<VectorElem>,
// što je podrazumevano operator > za VectorElem,
// što se u ovom primeru svodi na operator > za int.
sort(vektorBrojeva.begin(), vektorBrojeva.end(), greater<VectorElem>());
for_each(vektorBrojeva.begin(), vektorBrojeva.end(), writeElem);

return 0;
}
```

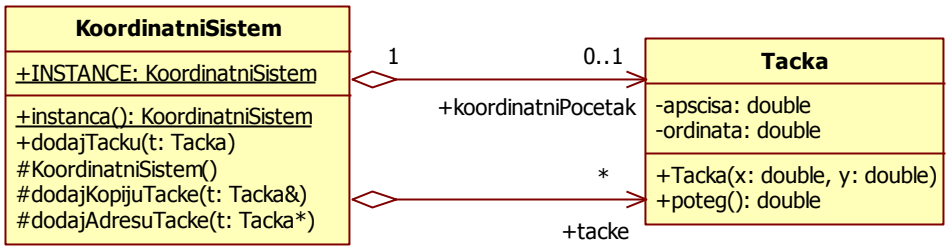
7. Projektni obrasci

7.1. Singleton DP

Ilustrovati na primeru primenu projektnog obrasca Singleton.

Rešenje:

UML dijagram klasa




```
// Copyright (C) 2008 ETF Mighty Coders

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_TACKA_4785573C00AB_INCLUDED
#define _INC_TACKA_4785573C00AB_INCLUDED

#include <ostream>

/// @brief Tačka u ravni predstavljena koordinatama.
///
/// Klasa je opremljena i operatorima < i ==,
/// da bi mogla biti upotrebljena kao element neke od STL zbirki.
/// Pored ovoga, klasa je opremljena i pomoćnim klasama
/// TackaLess i TackaGreater koje služe da obezbede mogućnost
/// da primerci klase mogu da budu lako upotrebljeni i u zbirkama
/// koje sadrže pokazivače na njih.
class Tacka
{
public:
/// @brief Podrazumevani ujedno konstruktor konverzije.
///
/// @param[in] pv_apscisa X koordinata, podrazumevano @c 0.
/// @param[in] pv_ordinata Y koordinata, podrazumevano @c 0.
Tacka(double pv_apscisa = 0.0, double pv_ordinata = 0.0);

/// Računa udaljenost tačke od koordinatnog početka.
double poteg() const;

/// Operator ispisa.
friend std::ostream& operator<<(std::ostream& o, const Tacka& rhs);

/// @brief Utvrđuje da li je data tačka manje udaljena od
/// koordinatnog početka od tekuće tačke.
///
/// @param[in] rhs Referenca na tačku sa kojom treba uporediti tekuću.
bool operator<(const Tacka& rhs) const;

/// @brief Utvrđuje da li je data tačka jednako udaljena od
/// koordinatnog početka kao i tekuća tačka.
```

```
///
/// @note Zbog mogućih grešaka pri zaokruživanju, u ovom primeru
/// dva double podatka smatraju se međusobno jednakim
/// ako je razlika između njih manja od @c 0.000001.
/// @param[in] rhs Referenca na tačku sa kojom treba uporediti tekuću.
bool operator==(const Tacka& rhs) const;

/// @brief Pomoćna klasa za poređenje dve tačke
/// po udaljenosti od koordinatnog početka.
///
/// Pošto je operator < preklopljen za objekte, a ne za pokazivače,
/// na ovaj način je obezbeđena podrška za uređivanje zbirki.
class TackaLess
{
public:
   /// @brief Funkcijski operator za poređenje dve tačke
   /// zadate preko pokazivača.
   ///
   /// @param[in] pp_leviOperand Pokazivač na levi operand za operator <.
   /// @param[in] pp_desniOperand Pokazivac na desni operand za operator <.
   /// @retval true Ako je tačka na koju pokazuje levi operand
   /// bliža koordinatnom početku od tačke na koju pokazuje desni operand.
   /// @retval false U ostalim slučajevima.
   bool operator () (Tacka* pp_leviOperand, Tacka* pp_desniOperand) const
   {
      return *pp_leviOperand < *pp_desniOperand;
   }
};

/// @brief Pomoćna klasa za poređenje dve tačke
/// po udaljenosti od koordinatnog početka.
///
/// Pošto su operatori < i == preklopljeni za objekte, a ne za pokazivače,
/// na ovaj način je obezbeđena podrška za uređivanje zbirki.
class TackaGreater
{
public:
   /// @brief Funkcijski operator za poređenje dve tačke
   /// zadate preko pokazivača.
   ///
   /// @param[in] pp_leviOperand Pokazivač na levi operand za operator <.
   /// @param[in] pp_desniOperand Pokazivac na desni operand za operator <.
   /// @retval true Ako je tačka na koju pokazuje levi operand
   /// dalja od koordinatnog početka nego tačka na koju pokazuje desni operand.
   /// @retval false U ostalim slučajevima.
   bool operator () (Tacka* pp_leviOp, Tacka* pp_desniOp) const
   {
      return !(*pp_leviOp < *pp_desniOp || *pp_leviOp == *pp_desniOp);
   }
};

private:
/// X koordinata tačke.
double apscisa;
/// Y koordinata tačke.
double ordinata;
};
#endif /* _INC_TACKA_4785573C00AB_INCLUDED */
// Copyright (C) 2008 ETF Mighty Coders
```

```
#include "Tacka.h"

#include <cmath> // Zbog funkcija sqrt i fabs
#include <iomanip> // Zbog manipulatora setprecision

Tacka::Tacka(double pv_apscisa, double pv_ordinata)
: apscisa(pv_apscisa), ordinata(pv_ordinata) { }

double Tacka::poteg() const
{
return sqrt(apscisa * apscisa + ordinata * ordinata);
}

std::ostream& operator<<(std::ostream& o, const Tacka& rhs)
{
return o  << std::fixed << std::setprecision(3) << rhs.poteg() << " - ("
          << rhs.apscisa << ',' << rhs.ordinata << ')' << std::endl;
}

bool Tacka::operator<(const Tacka& rhs) const
{
return poteg() < rhs.poteg();
}

bool Tacka::operator==(const Tacka& rhs) const
{
return fabs(poteg() - rhs.poteg()) < 0.000001;
}
```

Programski kod za klasu KoordinatniSistem

```
// Copyright (C) 2008 ETF Mighty Coders

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_KOORDINATNISISTEM_478556FD034B_INCLUDED
#define _INC_KOORDINATNISISTEM_478556FD034B_INCLUDED

#include <queue>

#include "Tacka.h"

/// @brief Dvodimenzionalni koordinatni sistem.
///
/// Pretpostavka je da je koordinatni sistem zadužen za vođenje evidencije
/// o postojećim tačkama, i njihovo uništavanje.
/// Postoji tačno jedan primerak koordinatnog sistema [Singleton DP].
/// @note Klasa čuva tačke u dve zbirke, kako bi bila naglašena razlika
/// između zbirki koje treba čuvaju adrese objekata i
/// zbirki koje čuvaju kopije objekata.
class KoordinatniSistem
{
public:
/// Adresa tekuće jedine instance klase [Singleton DP].
static KoordinatniSistem* Instance();

/// Destruktor uništava tačke sadržane preko pokazivača.
virtual ~KoordinatniSistem();
```

```
/// U nekim specifičnim slučajevima je potrebno uništiti instancu klase.
static void ReleaseInstance();

/// @brief Dodaje tačku u obe zbirke.
/// @param[in] pp_tacka Pokazivač na tačku koju treba dodati u zbirke.
void dodajTacku(Tacka *pp_tacka);

/// @brief Ispis tačaka sadržanih u obe zbirke.
///
/// @param[in] os Tok u koji će biti ispisani podaci o tačkama.
void ispisiTacke(std::ostream& os);

protected:
/// Zaštićeni podrazumevani konstruktor [Singleton DP].
KoordinatniSistem();

/// Briše sve sadržane tačke (prazni obe zbirke).
void obrisiSveTacke();

/// @brief Dodaje kopiju tačke u zbirku tačaka.
///
/// @param[in] pv_tacka Tačku koju treba dodati u zbirku.
void dodajKopijuTacke(Tacka pv_tacka);

/// @brief Dodaje adresu tačke u zbirku adresa tačaka.
///
/// @param[in] pp_tacka Pokazivač na tačku koju treba dodati u zbirku.
void dodajAdresuTacke(Tacka *pp_tacka);

private:
/// @brief Prioritetni red za čekanje koji čuva kopije objekata.
///
/// Podrazumevana zbirka u kojoj će biti čuvane kopije je @c std::vector.
/// Podrazumevano uređenje reda je @c less<Tacka>,
/// tj. uređenje prema operatoru @c <.
std::priority_queue<Tacka> kopijeTacaka;

/// @brief Prioritetni red za čekanje koji čuva adrese objekata.
///
/// Zbirka u kojoj će biti čuvane adrese je @c std::vector<Tacka*>.
/// Uređenje reda je @c Tacka::TackaGreater, što je pomoćna klasa
/// za uređenje reda prema opadajućoj udalenosti od koordinatnog početka.
std::priority_queue<Tacka*, std::vector<Tacka*>, Tacka::TackaGreater>  adreseTacaka;

/// Adresa tekuće jedine instance klase [Singleton DP].
static KoordinatniSistem* instance;

/// Koordinatni početak.
static const Tacka koordinatniPocetak;

};

#endif /* _INC_KOORDINATNISISTEM_478556FD034B_INCLUDED */
```

```
// Copyright (C) 2008 ETF Mighty Coders

#include "KoordinatniSistem.h"

#include <ostream>

KoordinatniSistem* KoordinatniSistem::Instance()
{
if (instance == 0)
  instance = new KoordinatniSistem();
return instance;
}

KoordinatniSistem::~KoordinatniSistem() { obrisiSveTacke(); }

void KoordinatniSistem::ReleaseInstance()
{
if (instance != 0)
    delete instance, instance = 0;
}

void KoordinatniSistem::dodajTacku(Tacka *pp_tacka)
{
dodajKopijuTacke(*pp_tacka);
dodajAdresuTacke(pp_tacka);
}

void KoordinatniSistem::ispisiTacke(std::ostream &os)
{
// Prilikom ispisa zbirke kopija,
// Potrebno je samo izbaciti ispisanu tačku sa prvog mesta u redu.
os << "Kopije unetih tacaka (iz zbirke kopija): " << std::endl;
while (!kopijeTacaka.empty())
{
  os << kopijeTacaka.top();
  kopijeTacaka.pop();
}
// Prilikom ispisa zbirke adresa,
// potrebno je i dealocirati objekat koji napušta red.
os << "Unete tacke (iz zbirke adresa): " << std::endl;
while (!adreseTacaka.empty())
{
  Tacka *pTacka = adreseTacaka.top();
  os << *pTacka;
  adreseTacaka.pop();
  delete pTacka;
  // Pošto MSVC u Debug načinu rada proverava za svaku metodu STL zbirke
  // da li pristupa do ispravno alocirane dinamičke memorije,
  // sledeći programski kod je komentarisan
  // (iako je logički ispravan i funkcionalno identičan prethodnom)
  // radi izbegavanja poruke "Debug assertion failed... invalid heap".
//   os << adreseTacaka.top();
//   delete adreseTacaka.top();
//   adreseTacaka.pop();
}
}

KoordinatniSistem::KoordinatniSistem() { }
```

```
void KoordinatniSistem::obrisiSveTacke()
{
// Kod kopija ne treba raditi ništa posebno, samo isprazniti zbirku.
while (!kopijeTacaka.empty())
   kopijeTacaka.pop();

// Kod adresa treba i dealocirati svaku tačku.
while (!adreseTacaka.empty())
{
   Tacka *pTacka = adreseTacaka.top();
   adreseTacaka.pop();
   delete pTacka; // Po
   // Popogledati komentar u funkciji ispisiTacke();
   // delete adreseTacaka.top();
   // adreseTacaka.pop();
}
}

void KoordinatniSistem::dodajKopijuTacke(Tacka pv_tacka)
{
kopijeTacaka.push(pv_tacka);
}

void KoordinatniSistem::dodajAdresuTacke(Tacka *pp_tacka)
{
adreseTacaka.push(pp_tacka);
}

KoordinatniSistem* KoordinatniSistem::instance = 0;

const Tacka KoordinatniSistem::koordinatniPocetak;
```

7.2. Singleton DP - Kompletan primer.

Rešenje:

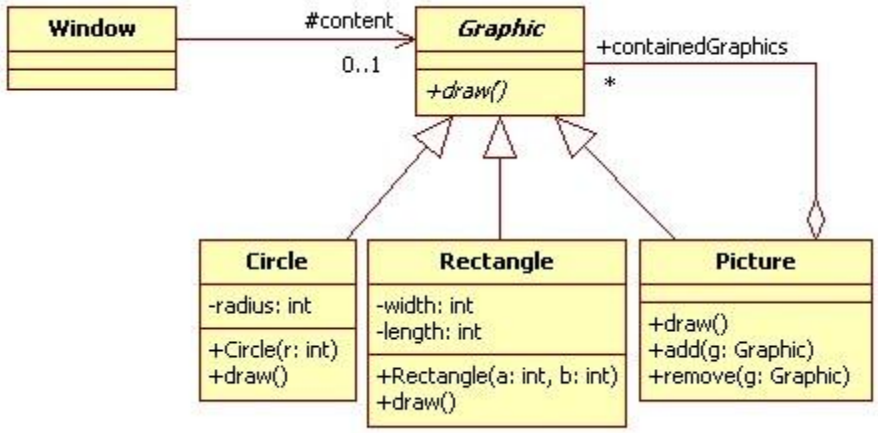
```
/*
 * model.h
 *
 *  Created on: 25.12.2015.
 *      Author: Nemanja Kojic
 */

#ifndef MODEL_H_
#define MODEL_H_

/**
 * Class Model implemented according to Singleton DP.
 * Dynamic singleton object.
 */
#include <vector>
class ModelElement;
using namespace std;

class Model {
public:
   // Operations that facilitate Singleton DP.
   static Model* get();
   static void release() { delete instance; instance = nullptr; }

   // Interface of Model.
   void add(ModelElement* e) { elements.push_back(e); }
   void clear() { elements.clear(); }

protected:
   // Protected members - control model object instantiation/deletion.
   Model() {}
   virtual ~Model() { elements.clear(); }

private:
   vector<ModelElement*> elements;
   static Model* instance;

private: // Disabled member functions.
   Model(const Model& m) =delete;
   Model& operator=(const Model& m) =delete;
   Model(Model&&) =delete;
   Model& operator=(Model&& m) =delete;
};

// Dummy class
class ModelElement {};
#endif /* MODEL_H_ */
```

```cpp
/*
 * model.cpp
 *
 *  Created on: 25.12.2015.
 *      Author: Nemanja Kojic
 */

#include "model.h"

// Allocate static pointer and init to nullptr.
Model* Model::instance = nullptr;

Model* Model::get() {
   if (instance == nullptr) {
      instance = new Model();
   }
   return instance;
}


/*
 * mode_main.cpp
 *
 *  Created on: 25.12.2015.
 *      Author: Nemanja Kojic
 */

#include "model.h"

// void f0(Model m) { }
void f1(Model& m) { }
void f2(Model&& m) { }

/** Disabled features are commented out. */
int main() {
   Model::get()->add(new ModelElement);
   // Model m1;
   // Model m2(*Model::get());
   // Model m3(std::move(*Model::get()));
   // m2 = *Model::get();
   // f0(*Model::get());
   // f0(std::move(*Model::get()));

   // It is possible to pass reference to the object.
   f1(*Model::get());
   f2(std::move(*Model::get()));
   // delete Model::get();
   return 0;
}
```

7.3. Projektni obrazac Composite

Grafički element (*Graphic*) može da se iscrta. Krug (*Circle*) je grafički element koji ima zadat poluprečnik i iscrtava se kao istoimena geometrijska figura. Pravougaonik (*Rectangle*) je grafički element koji ima zadate stranice i iscrtava se kao istoimena geometrijska figura. Slika (*Picture*) je složeni grafički element koji sadrži druge grafičke elemente. Stvara se prazna nakon čega joj je moguće dodati grafički element ili ukloniti zadati grafički element. Slika se iscrtava tako što poziva iscrtavanje sadržanih grafičkih elemenata. Implementirati navedeni problem korišćenjem projektnog obrasca *Composite*.

Rešenje:



Slika 1 Model klasa sistema. Projektni obrazac Composite.

```
/*
 * composite.h
 *
 *  Created on: 18.12.2014.
 *      Author: Nemanja Kojic
 */

#ifndef COMPOSITE_H_
#define COMPOSITE_H_

class Graphic
{
public:
// Pure virtual (abstract) operation that draws a graphic.
virtual void draw() =0;

protected:
static int nestingLevel;
};

class Circle : public Graphic
{
int r;
public:
Circle(int r): r(r) { }

void draw();
};
```

```
class Rectangle : public Graphic
{
int a, b;
public:
Rectangle(int a, int b) : a(a), b(b) { }

void draw();
};

#include <vector>
using namespace std;

class Picture : public Graphic
{
// The association end "containedGraphics".
// Implemented as a vector of pointers to the abstract class objects.
std::vector<Graphic*> containedGraphics;

public:
// Draws the composite picture.
void draw();

// Adds the given abstract graphic to the picture.
void add(Graphic* g) { containedGraphics.push_back(g); }

// Removes the given graphic g from the picture.
void remove(Graphic* g);

Graphic* getGraphic(int index)
  { return containedGraphics.at(index); }

const std::vector<Graphic*>& getChildren() const
  { return containedGraphics; }
};
#endif /* COMPOSITE_H_ */
```

```
/*
 * composite.cpp
 *      Author: Nemanja Kojic
 */
#include "composite.h"

#include <algorithm>
#include <iomanip>
#include <iostream>
using namespace std;

int Graphic::nestingLevel = 0;

void Picture::remove(Graphic* g)
{
auto position = std::find(containedGraphics.begin(), containedGraphics.end(), g);
// == vector.end() means the element was not found
if (position != containedGraphics.end())
   containedGraphics.erase(position);
}

void Circle::draw()
{
```

```
cout << setw(nestingLevel) << right
   << "C(" << r << ")" << endl;
}

void Rectangle::draw()
{
cout << setw(nestingLevel) << right
   << "R(" << a << b << ")" << endl;
}

void Picture::draw()
{
cout << setw(nestingLevel) << right << "{ "<< endl;
nestingLevel+=5;
for (auto it=containedGraphics.begin(); it != containedGraphics.end(); ++it)
{
   (*it)->draw();
}
nestingLevel-=5;
cout << setw(nestingLevel) << right << "} "<< endl;
}

/*
 * composite_main.cpp
 *      Author: Nemanja Kojic
 */
#include "composite.h"

int main()
{
Graphic* t1 = new Circle(1);
Graphic* t2 = new Circle(3);
Graphic* r1 = new Rectangle(1, 2);
Graphic* r2 = new Rectangle(3, 4);

Picture *picture = new Picture();
   picture->add(t1);
   Picture* p2 = new Picture();
      p2->add(t2);
      p2->add(r2);

   picture->add(p2);
   picture->add(r1);

picture->draw();

// Remove t1;
picture->remove(t1);
picture->draw();

// How to remove t2?
picture->remove(t2); // ?
picture->draw();

return 0;
}
```

7.4. Biblioteka (Composite DP, skica)

Biblioteka (Library) se sastoji od više *odeljaka* (*Department*). Svaki *odeljak* sadrži proizvoljno mnogo knjiga (*Book*), pri čemu se jedna knjiga može naći u više odeljaka. Odeljci se organizuju po žanru (*DeptByGenre*) ili po autoru (*DeptByAuthor*), pri čemu odeljci koji se organizuju po žanru mogu da imaju pododeljke (po žanru ili po autoru). Knjiga se dodaje u biblioteku pozivanjem odgovarajuće operacije klase **`Library`** koja pravi objekat tipa **`Book`** i ovaj objekat smešta u odgovarajuće odeljke na osnovu žanra i autora. Parametri ove operacije su: ime knjige, ime autora i naziv žanra. Zadaje se puno ime žanra, pri čemu se u zadatoj reči polazi od osnovnog žanra, a ime svakog sledećeg podžanra je razdvojeno znakom za razdvajanje (tačkom). Na primer, puno ime žanra može biti: **`"Umetnost.Likovna umetnost.Grafika"`**.

Postupak za smeštanje knjige je sledeći: Prolazi se kroz celu hijerarhiju odeljaka i njihovih pododeljaka i ispituju im se nazivi. Ukoliko je odeljak organizovan po imenu autora, a ime autora knjige odgovara imenu odeljka, knjiga se smešta u taj odeljak. Ukoliko je odeljak organizovan po žanru, ispituje se da li ime odeljka odgovara prvoj reči u zadatom punom imenu žanra. Ukoliko odgovara, prelazi se na ispitivanje pododeljaka, pri čemu se od zadatog imena žanra oduzima prva reč i tako dobijeno ime se dalje pretražuje. Ukoliko se knjiga ne smesti ni u jedan pododeljak, ona se smešta u početni odeljak. Ukoliko se u celoj biblioteci ne pronađe ni jedan odeljak u koji se može smestiti knjiga, operacija objekta biblioteka treba da vrati **`false`**. U suprotnom vraća **`true`**.

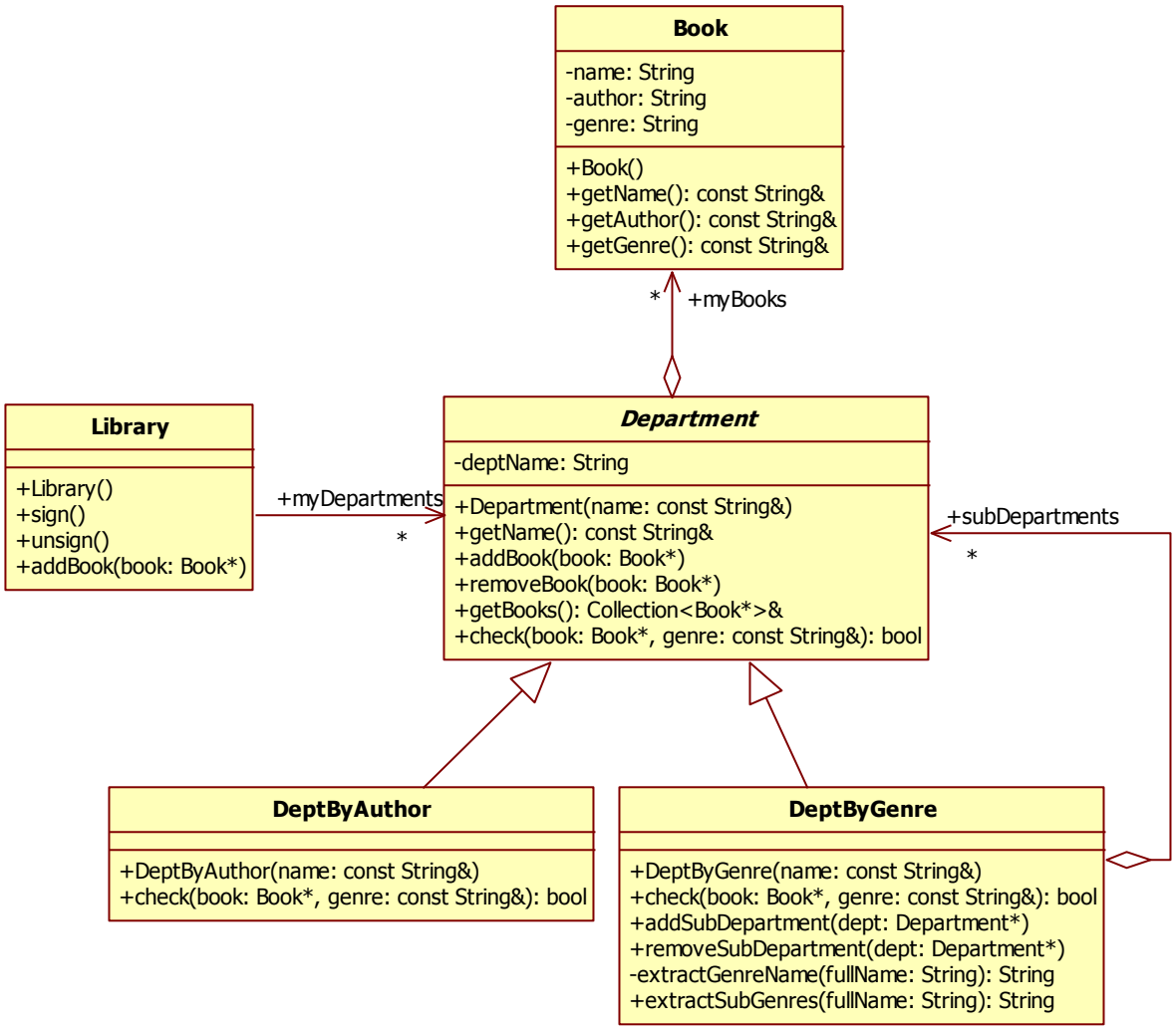
Napisati na programskom jeziku C++ klase za opisane koncepte. Napisati glavni program koji napravi biblioteku, dodaje joj odeljke i zatim testira stavljanje knjiga u biblioteku.

Rešenje:

Najpre će biti formiran model sistema, na osnovu kojeg može da se jako dobro sagleda kompleksnost sistema, definiše interfejs koncepata, struktura i odnosi (relacije). Model je napravljen pomoću jezika za modelovanje UML.

Kada se završi proces modelovanja, prelazi se na implementaciju. Danas postoji dosta alata koji mogu na osnovu UML model da izgenerišu kostur sistema/klasa, tako da programeri mogu da se bave isključivo implementacijom ključnih funkcionalnosti i algoritama, dok će generator koda za njih da generiše standardni šablonski kod. Danas u informatici postoji jedna posebna oblast istraživanja i premene naučnih dostignuća na modelovanja softvera i pravljenje softverskih sistema na osnovu modela koja se zove Model-based Engineering.

UML model softverskog sistema:



Implementacija klasa na programskom jeziku C++

```
class Book {
public:
Book(const String& aName, const String& aAuthor, const String& aGenre)
: name(aName), author(aAuthor), genre(aGenre) {}

const String& getName() { return name; }
const String& getAuthor() { return author; }
const String& getGenre() { return genre; }

private:
String name;
String author;
String genre;
};
```

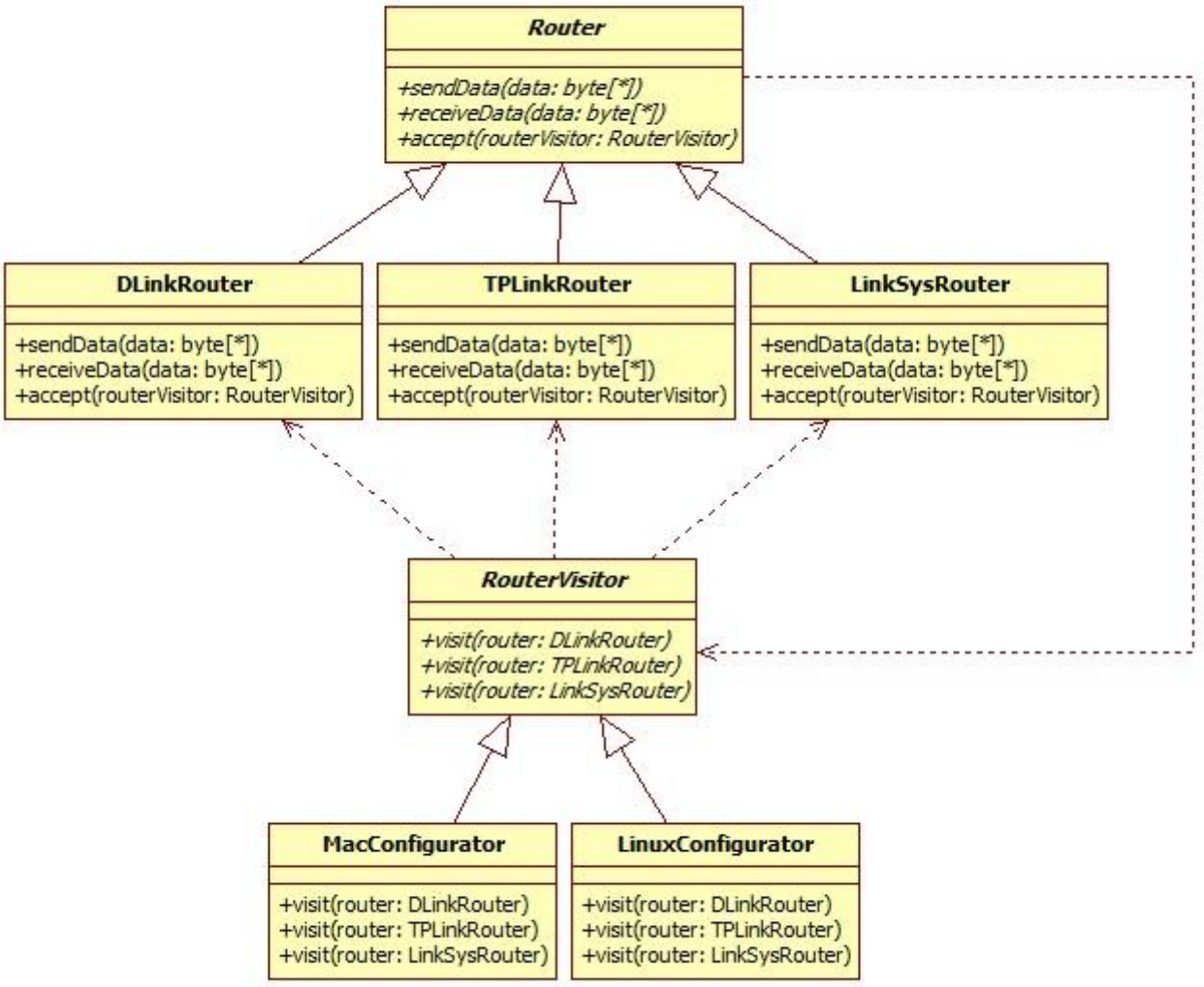
```
bool operator== (const Book& left, const Book& right) {
return (left.getName() == right.getName())
     && (left.getAuthor() == right.getAuthor())
     && (left.getGenre() == right.getGenre());
}

#include "Book.h"
class Department {
public:
Department(const String& name) : deptName(name) {}
const String& getName() { return deptName; }
void addBook(Book* book) { books.add(new Book(*book); }
Collection<Book*>& getBooks() { return books; }

void removeBook(Book* book) { books.remove(book); }
bool check(Book* book, const String& genre)=0;
private:
Collection<Book*> books;
String deptName;
};

#include "Department.h"
class DeptByAuthor : public Department {
public:
void DeptByAuthor(const String& name) : Department(name)
{ Library::instance()->sign(this); }

bool check(Book* book, const String& genre);
};

#include "Department.h"
class DeptByGenre : public Department {
public:
void DeptByGenre(const String& name) : Department(name) {}
bool check(Book* book, const String& genre);
private:
String extractGenreName(String fullName);
String extractSubGenres(String fullName);
Collection<Department*> subDepartments;
};

String extractGenreName(String fullName) {
// ... implement this method
}

String extractSubGenres(String fullName) {
// ... implement this method
}

#include "Department.h"
class Library {
public:
void Library();
void sign(Department *dept) { myDepartments.add(dept); }
void unsign(Department *dept) { myDepartments.remove(dept); }
void addBook(const String& name, const String& author, const String& genre);
private:
Collection<Department*> myDepartments;
};
```

7.5. Zadatak

Projektovati sistem klasa za fleksibilno konfigurisanje mreže rutera. Ruter može da prihvati i pošalje podatke. Postoje tri vrste rutera od sledećih proizvođača: TPLinkRouter, DLinkRouter, LinkSysRouter. Ruteri mogu da se konfigurišu za Mac ili Linux operativni sistem, a u planu je podrška i za druge operativne sisteme. Projektovati sistema klasa koji omogućava laku proširivost za nove platforme.

Rešenje:

Model sistema klasa



Implementacija klasa

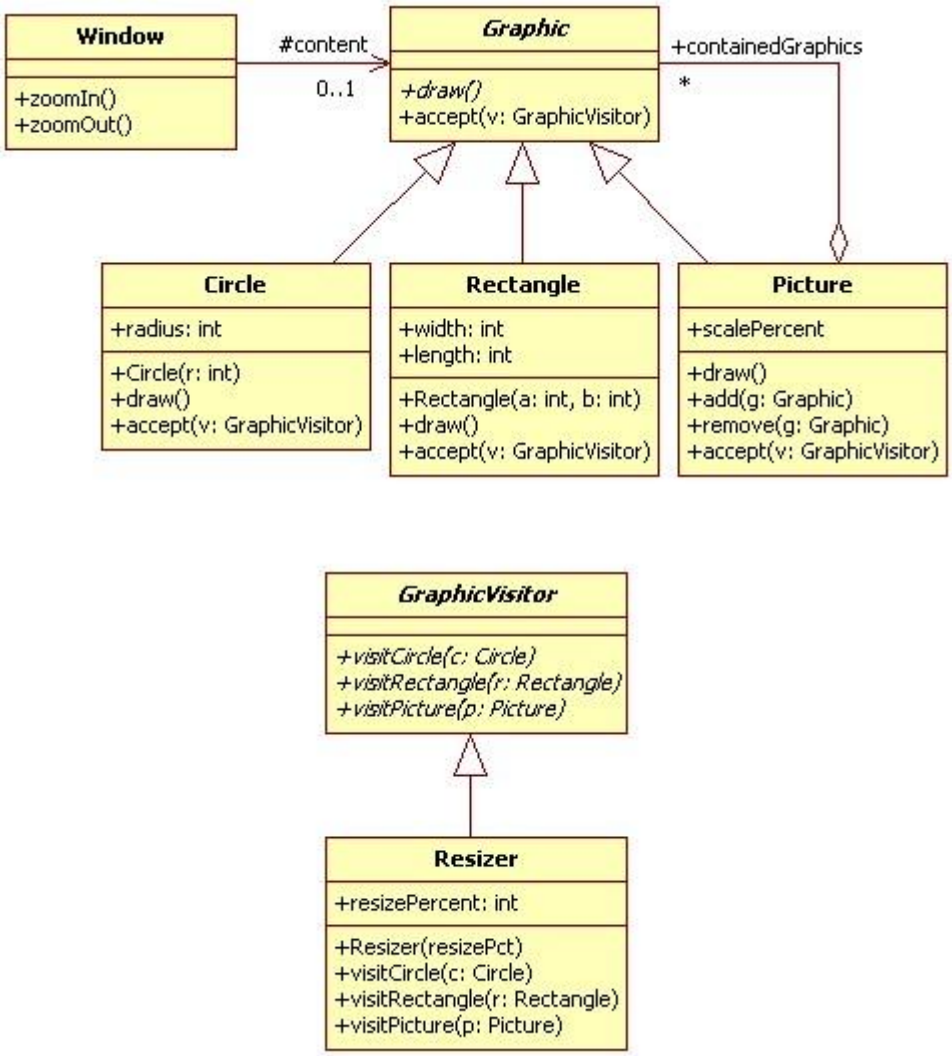
```
typedef char byte;
class Router {
public:
virtual void sendData(byte *data) = 0;
virtual void receiveData(byte *data) = 0;
virtual void accept(RouterVisitor *routerVisitor) = 0;
};

class TPLinkRouter;
class DLinkRouter;
class LinkSysRouter;
```

```
class RouterVisitor {
public:
virtual void visit(DLinkRouter *router) = 0;
virtual void visit(TPLinkRouter *router) = 0;
virtual void visit(LinkSysRouter *router) = 0;
};

class TPLinkRouter : public Router {
public:
void sendData(byte *data) { /*...*/ }
void receiveData(byte *data) { /*...*/ }
void accept(RouterVisitor *routerVisitor) { routerVisitor->visit(this); }
};

class LinkSysRouter : public Router {
public:
void sendData(byte *data) { /*...*/ }
void receiveData(byte *data) { /*...*/ }
void accept(RouterVisitor *routerVisitor) { routerVisitor->visit(this); }
};

class DLinkRouter : public Router {
public:
void sendData(byte *data) { /*...*/ }
void receiveData(byte *data) { /*...*/ }
void accept(RouterVisitor *routerVisitor) { routerVisitor->visit(this); }
};

#include <iostream>
using namespace std;

class MacConfigurator : public RouterVisitor {
public:
void visit(DLinkRouter *router)
{ cout << "DLinkRouter configuration for Mac complete." << endl; }

void visit(TPLinkRouter *router)
{ cout << "TPLinkRouter configuration for Mac complete." << endl; }

void visit(LinkSysRouter *router)
{ cout << "LinkSysRouter configuration for Mac complete." << endl; }
};

class LinuxConfigurator : public RouterVisitor {
public:
void visit(DLinkRouter *router)
{ cout << "DLinkRouter configuration for Linux complete." << endl; }

void visit(TPLinkRouter *router)
{ cout << "TPLinkRouter configuration for Linux complete." << endl; }

void visit(LinkSysRouter *router)
{ cout << "LinkSysRouter configuration for Linux complete." << endl; }
};
```

7.6. Skaliranje slike, projektni obrazac Posetilac (Visitor).

Dopuniti klase iz zadatka 7.2. dodavanjem boje grafičkim elementima. Grafički element ima boju i može da prihvatni objekat klase *GraphicVisitor*. GraphicVisitor može da poseti krug, pravouganik i sliku. Painter je posetilac (*GraphicVisitor*) koji menja boju svim figurama. Proširiti klase iz zadatka 7.2. implementacijom projektnog obrasca Visitor.

Rešenje:



```
/*
 * composite.h
 *      Author: Nemanja Kojic
 */

#ifndef COMPOSITE_H_
#define COMPOSITE_H_

class GraphicVisitor;

class Graphic
{
public:
// Pure virtual (abstract) operation that draws a graphic.
virtual void draw() =0;

virtual void accept(GraphicVisitor* v) =0;

protected:
static int nestingLevel;
};

class Circle : public Graphic
{
int r;
public:
Circle(int r): r(r) { }

void draw();
void accept(GraphicVisitor* v);

int getRadius() const { return r; }
void setRadius(int newRadius) { r = newRadius; }
};

class Rectangle : public Graphic
{
int a, b;
public:
Rectangle(int a, int b): a(a), b(b) { }

void draw();
void accept(GraphicVisitor* v);

void setLength(int newLength) { a = newLength; }
int getLength() const { return a; }

void setWidth(int newWidth) { b = newWidth; }
int getWidth() const { return b; }
};
```

```
#include <vector>
using namespace std;

class Picture : public Graphic
{
// The association end "containedGraphics".
// Implemented as a vector of pointers to the abstract class objects.
std::vector<Graphic*> containedGraphics;

int scalePercent = 100;
public:
// Draws the composite picture.
void draw();
void accept(GraphicVisitor* v);

// Adds the given abstract graphic to the picture.
void add(Graphic* g) { containedGraphics.push_back(g); }
// Removes the given graphic g from the picture.
void remove(Graphic* g);

Graphic* getGraphic(int index)
  { return containedGraphics.at(index); }
const std::vector<Graphic*>& getChildren() const
  { return containedGraphics; }

void setScalePercent(int scalePercent) { this->scalePercent = scalePercent; }
int getScalePercent() const { return scalePercent; }
};

class GraphicVisitor
{
public:
virtual void visitCircle(Circle* c)=0;
virtual void visitRectangle(Rectangle* c)=0;
virtual void visitPicture(Picture* c)=0;
};
class Resizer : public GraphicVisitor
{
int resizePercent;
public:
Resizer(int resizePercent) { this->resizePercent = resizePercent; }

virtual void visitCircle(Circle* c)
{
   c->setRadius(c->getRadius() * (1 + resizePercent / 100.));
}
virtual void visitRectangle(Rectangle* r)
{
   r->setLength(r->getLength() * (1 + resizePercent / 100.));
   r->setWidth(r->getWidth() * (1 + resizePercent / 100.));
}
virtual void visitPicture(Picture* c)
{
   c->setScalePercent(100 + resizePercent);
}
};

#endif /* COMPOSITE_H_ */
```

```
/*
 * composite.cpp
 *      Author: Nemanja Kojic
 */

#include "composite.h"

#include <algorithm>
#include <iomanip>
#include <iostream>
using namespace std;

int Graphic::nestingLevel = 0;

void Picture::remove(Graphic* g)
{
auto position = std::find(containedGraphics.begin(), containedGraphics.end(), g);
// == vector.end() means the element was not found
if (position != containedGraphics.end())
   containedGraphics.erase(position);
}

void Circle::draw()
{
cout << setw(nestingLevel) << right
   << "C(" << r << ")"
   << endl;
}

void Rectangle::draw()
{
cout << setw(nestingLevel) << right
   << "R(" << a << "," << b << ")"
   << endl;
}

void Picture::draw()
{
cout << setw(nestingLevel) << right << "{ "<< endl;
nestingLevel+=5;
for (auto it=containedGraphics.begin(); it != containedGraphics.end(); ++it)
{
   (*it)->draw();
}
nestingLevel-=5;
cout << setw(nestingLevel) << right << "} "<< endl;
}

void Circle::accept(GraphicVisitor* v)
{
v->visitCircle(this);
}

void Rectangle::accept(GraphicVisitor* v)
{
v->visitRectangle(this);
}
```

```
   void Picture::accept(GraphicVisitor* v)
   {
   vector<Graphic*>& g = containedGraphics;
   for (auto it = g.begin(); it != g.end(); ++it)
   {
      (*it)->accept(v);
   }
   v->visitPicture(this);
}


   /*
    * composite_main.cpp
    *
    *  Created on: 18.12.2014.
    *      Author: Nemanja Kojic
    */

   #include "composite.h"

   void zoomIn(Graphic *g) {
   Resizer r(+25);
   g->accept(&r);
   g->draw();
   }

   void zoomOut(Graphic *g) {
   Resizer r(-25);
   g->accept(&r);
   g->draw();
   }

   int main()
   {
   Graphic* t1 = new Circle(10);
   Graphic* t2 = new Circle(30);
   Graphic* r1 = new Rectangle(10, 20);
   Graphic* r2 = new Rectangle(30, 40);

   Picture *picture = new Picture();
   {
      picture->add(t1);
      Picture* p2 = new Picture();
      {
         p2->add(t2);
         p2->add(r2);
      }
      picture->add(p2);
      picture->add(r1);
   }
   picture->draw();

   zoomIn(picture);
   zoomIn(picture);
   zoomOut(picture);
   return 0;
   }
```

7.7. Projektni obrazac Strategija (Strategy DP, uređivanje teksta)

Uređivač teksta (*TextFormatter*) može da uredi tekst. Uređivanje teksta se radi tako što se prvo isprave eventualne sintaksne greške, a zatim se tekst poravna na zadati način. Moguće je tekst poravnati na tri načina: poravnanje levo (*LeftAlignment*), desno (*RightAlignment*), centrirano (*CenterAlignment*). Poravnanje (*AlignmentStrategy*) se može promeniti u vreme izvršavanja programa. Tekst se čita iz fajla liniju po liniju i svaka linija se poravnava na zadati način. Na kraju obrade, ulazni fajl je potrebno zatvoriti. Napisati na C++ glavni program, koji napravi objekat uređivača teksta, zatim od korisnika zahteva izbor željenog formatiranja, a zatim ispiše dobijeni tekst na standardni izlaz. Program se izvršava sve dok korisnik ne unese vrednost 0.

Rešenje:

Ovaj problem se može rešiti upotrebom dva projektna uzorka: šablonski metod i strategija. Šablonske metode se metode koje imaju fiksno definisan niz koraka koji moraju da budu odrađeni u okviru kompleksnije obrade. U ovom primeru, šablonski su specificirane: (1) metoda u klasi TextFormatter koja prvo proverava sintaksne greške, a zatim poravnava tekst i (2) metoda u samoj klasi Alignment koja otvori fajl, čita liniju po liniju i svaku poravnava i zatim zatvori fajla. Poravnanje teksta u odnosu na uređivač teksta predstavlja strategiju.

```
// Purpose.  Strategy design pattern demo
//
// Discussion.  The Strategy pattern suggests: encapsulating an algorithm
// in a class hierarchy, having clients of that algorithm hold a pointer
// to the base class of that hierarchy, and delegating all requests for
// the algorithm to that "anonymous" contained object.  In this example,
// the Strategy base class knows how to collect a paragraph of input and
// implement the skeleton of the "format" algorithm.  It defers some
// details of each individual algorithm to the "justify" member which is
// supplied by each concrete derived class of Strategy.  The TestFormatter class
// models an application class that would like to leverage the services of
// a run-time-specified derived "Strategy" object.
// http://www.vincehuston.org/dp/StrategyDemosCpp

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class AlignmentStrategy;

class TextFormatter {
public:
enum StrategyType { Dummy, Left, Right, Center };
TextFormatter() { strategy_ = NULL; }
void setAlignmentStrategy( int type, int width );
void formatText();
private:
virtual void correctSyntaxErrors()
{
   /*
     Apply syntactic rules...
   */
}
AlignmentStrategy*  strategy_;
};
```

```
class AlignmentStrategy {
public:
AlignmentStrategy( int width ) : width_( width ) { }

void alignText ()
{
   char line[80], word[30];
   ifstream  inFile( "quote.txt", ios::in );
   line[0] = '\0';

   inFile >> word;
   strcat( line, word );
   while (inFile >> word)
   {
      if (strlen(line) + strlen(word) + 1 > width_)
        justify( line );
      else
        strcat( line, " " );
      strcat( line, word );
   }
   justify( line );
}

protected:
int     width_;
private:
virtual void justify( char* line ) = 0;
};

class LeftStrategy : public AlignmentStrategy {
public:
LeftStrategy( int width ) : AlignmentStrategy( width ) { }
private:
void justify( char* line )
{
   cout << line << endl;
   line[0] = '\0';
}
};

class RightStrategy : public AlignmentStrategy {
public:
RightStrategy( int width ) : AlignmentStrategy( width ) { }
private:
void justify( char* line )
{
   char  buf[80];
   int   offset = width_ - strlen( line );
   memset( buf, ' ', 80 );
   strcpy( &(buf[offset]), line );
   cout << buf << endl;
   line[0] = '\0';
}
};
```

```
class CenterStrategy : public AlignmentStrategy
{
public:
CenterStrategy( int width ) : AlignmentStrategy( width ) { }
private:
void justify( char* line )
{
   char  buf[80];
   int   offset = (width_ - strlen( line )) / 2;
   memset( buf, ' ', 80 );
   strcpy( &(buf[offset]), line );
   cout << buf << endl;
   line[0] = '\0';
}
};

void TextFormatter::setAlignmentStrategy( int type, int width )
{
// What if someone enters an invalid type value?
if (strategy_) delete strategy_;
if      (type == Left)   strategy_ = new LeftStrategy( width );
else if (type == Right)  strategy_ = new RightStrategy( width );
else if (type == Center) strategy_ = new CenterStrategy( width );
}

// Template method.
void TextFormatter::formatText()
{
correctSyntaxErrors();
strategy_->alignText();
}

int main()
{
TextFormatter test;
int      answer, width;
cout << "Exit(0) Left(1) Right(2) Center(3): ";
cin >> answer;
while (answer)
{
   cout << "Width: ";
   cin >> width;
   test.setAlignmentStrategy( answer, width );
   test.formatText();
   cout << "Exit(0) Left(1) Right(2) Center(3): ";
   cin >> answer;
}
return 0;
}
```

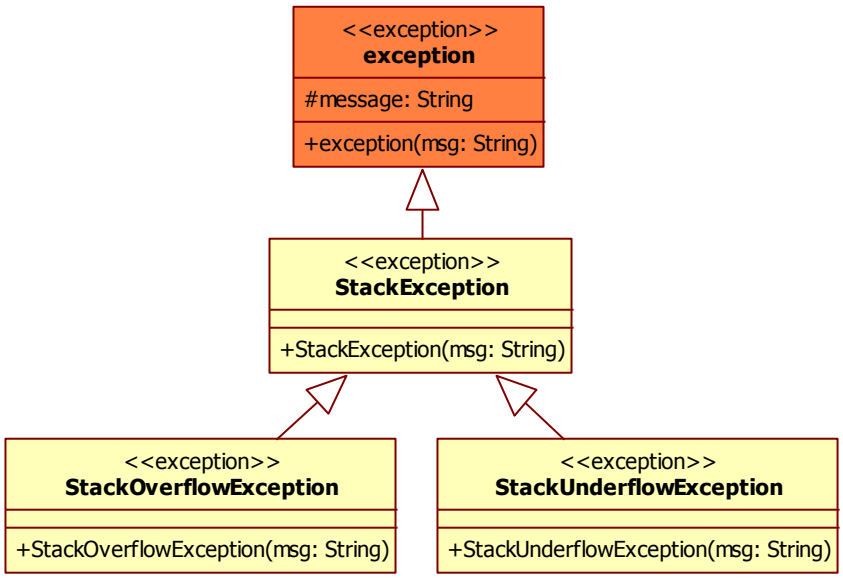
8. Obrada grešaka u programima – mehanizam izuzetaka

8.1. Stek (upotreba izuzetaka za obradu grešaka)

Napisati na programskom jeziku C++ program koji ilustruje obradu izuzetaka na primeru klase za stek.

Rešenje:

UML model izuzetaka




```
// stack.h
#include <exception>
#include <iostream>
#include <vector>

using namespace std;

// Exception hierarchy is formed by inheriting from the base
// class exception.
class StackException : public exception
{
public:
   StackException(const char* msg): exception(msg) { }
};

class StackOverflowException : public StackException
{
public:
   StackOverflowException(const char* errorDesc) : StackException(errorDesc) {}
};


class StackUnderflowException : public StackException
{
public:
   StackUnderflowException(const char* errorDesc) : StackException(errorDesc) { }
};
```

```
class Stack {
   int capacity;
   vector<int> content;
public:
   Stack(int cap) :capacity(cap) {  }
   // HIGHLY RECOMMENDED:
   //    DO    NOT    add    the    dynamic    throw    list    to    a    method's    signature.
   // E.g. void Stack::push(int val) throw(StackOverflowException);.
   // That's why the methods of class Stack do not declare exceptions they can throw.
   void Stack::push(int val);
   int  Stack::pop();
};


// stack.cpp
void Stack::push(int val)
{
   if (content.size() == capacity)
      throw StackOverflowException("Stack was full, when push was called!");
   content.push_back(val);
}

int Stack::pop()
{
   if (content.size() == 0)
      throw StackUnderflowException("Stack was empty, when pop was called.");
   int result = content.back(); // retrieves value of the last element in the vector.
   content.pop_back(); // removes the last element from the vector.
   return result;
}

void testStack(Stack& s){
   cout << "PUSH 2" << endl; s.push(2);
   cout << "PUSH 3" << endl; s.push(3);
   cout << "PUSH 4" << endl; s.push(4);
   cout<<"POP =" << s.pop() << endl;
   cout<<"POP =" << s.pop() << endl;
   cout<<"POP =" << s.pop() << endl;
   // this statement will throw StackUnderflowException.
   cout<<"POP =" << s.pop() << endl;
}

void main()
{
   Stack s(10);
   try {
      testStack(s); // when this method fails, the rest of the try block is skipped.
      testStack(s);
   }
   catch(StackUnderflowException& se) {
      cout<<se.what() <<endl;
   }
   catch(StackException& se) {
      cout<<se.what() <<endl;
   }
   catch(...) { // The most general catch handler - can catch all throwable values.
      cout<<se.what() <<endl;
   }
   cout<<”END”<<endl;
}
```

Kratak osvrt na mehanizam izuzetaka i preporuke za njhovo korišćenje:

Izuzeci omogućavaju prijavljivanje grešaka u toku izvršavanja program i reagovanje na "izuzetne" situacije. Kada se u programu "podigne" ili "baci" (throw) izuzetak, program logički napušta tok kontrole koji je bio aktuelna do pojave greške i prelazi u režim oporavka od greške.

Uopšteno govoreći, svaki netrivijalan C++ program, čije izvršavanje započinje pozivom main funkcije, sastoji se od mnoštva potprograma, koji se mogu pozivati direktno iz main funkcije, ili pak se programi mogu međusobno pozivati sa proizvoljnom dubinom ugneždavanja poziva. Svi ugneždeni pozivi čuvaju se na programskom steku, i kad se poslednje pozvana metoda ili funkcija okonča, sa steka poziva se skida njen aktivacioni zapis i vrši se povratak u konktekst pozivaoca. Tako se redom dešava za sve funkcije, pa i za main koja se nalazi na početku lanca ugneždenih poziva.

Zašto je ovo bitno za objašnjene izuzetaka? Pošto se C++ program čine izvršne naredbe koje se zapisuju u telima nekakvih funkcija ili metoda, jasno je da se bacanje izuzetka uvek dešava u trenutku kad postoji verovatno nekoliko ugneždenih poziva potprograma. Kada se izvrši naredba throw, prelazak programa u režim oporavka podrazumeva traženje najbližeg try-catch blocka u nekom okružujućem potprogramu pozivaocu, idući sa kraja lanca pozivanja, koji može da prihvati objekat bačenog izuzetka (postoji catch grana sa odgovarajućim tipom ili tipom koji je opštiji). Kada se takav try-catch blok pronađe, izvrši se catch grana i kontrola toka program se nastavlja iza try-catch bloka u kom je izuzetak uhvaćen i obrađen. Pošto se takav try-catch blok najverovatnije neće pronaći u potprogramu koji baca izuzetak (a i zašto bi, to nije preporučljivo), već u nekom okružujućem potprogramu pozivaocu, program dok traži try-catch blok u režimu oporavka radi režijski posao koji se zove "razmotavanje steka" (*stack unwinding*). Razmotavanje steka podrazumeva napuštanje opsega važenja unazad svakog pozvanog potprograma u lancu, koji ne može da obradi ili ne obrađuje dati izuzetak, pri čemu svako napuštanje opsega podrazumeva ono što se standardno dešava u takvim situacijama, a to je destrukcija lokalnih automatskih objekata. Razmotavanje steka, odnosno lanca pozvanih potprograma, prekida se kod onog koji je obradio izuzetak, nakon čega je program oporavljen od greške (ali sa eventualnim posledicama) i nastavlja se sa normalnim izvršavanjem.

Izuzeci, kao što i sam naziv kaže, koriste se za oporavak od **izuzetnih** situacija. Takve situacije nastaju ili usled neadekvatnog poštovanja nekog interfejsa (programer nije do kraja ispoštovao specifikaciju nekog API-a) ili prosto usled postojanja neke logičke greške u programu (na primer, programer nije dobro odradio validaciju ulaznih podataka, ili je zaboravio da čisti neki resurs ili strukturu podataka i td). Međutim, NIKAKO SE NE PREPORUČUJE korišćenje izuzetaka za signaliziranje povratne vrednosti nekog potprograma. Takve situacije su jako česte, uključuju dosta režijskog vremena za razmotavanje steka i odaju utisak LOŠE STRUKTURIRANOG rešenje. Za takve potprograme je bolja opcija da se koristi klasična povratna vrednost + if. Napomena: simuliranje if-then-else strukture pomoću try-catch bloka i izuzetaka može se okarakterisati kao jedna standardna početnička greška i anomalija u programu poznata pod nazivom "object spaghetti", što nije ništa drugo nego jedan fino upakovani GOTO.

NIKAKO OVAKO - dosta je cesto.	Preporuka za ovaj scenario.

`try {`
`  obj.method();`
`}`
`catch (ResultOK) { /* do something */  }`
`catch (ResultWrong) { /* react on error*/}`
	`int result = obj.method();`
	`if (result == ResultOK) { }`
	`else if (result == ResultWrong) { }`

Preporuke za deklarisanje izuzetaka i obradu.

Izuzeci koje neka metoda može da baci se mogu deklarisati u samom potpisu (dodavanjem klauzule throw(...)), ali se to više ne preporučuje.Na jeziku C++ svi izuzeci su neproveravani (***unchecked***), što znači da komajler neće proveriti niti opmenuti programera da njegova metoda može da baca i neke izuzetke koji nisu deklarisani. Isto tako, kompajler neće proveriti da li programer obrađuje ili prosleđuje i druge izuzetke koje mogu da bace druge metode koje on poziva u implementaciji svoje. Pošto sam kompajler ne proverava izuzetke, onda se i ne preporučuje njihovo navođenje u throw listi iz dva razloga: ne mora se brinuti o konceptu koji sam kompajler ne podržava, a sa druge strane to može imati određene implikacije na samu kontrolu toka programa, kao i oporavka od greške.

Slučaj upotrebe 1: **`void transferData(byte[] data);`**

Ova hipotetička procedura ili metoda ne deklariše nijedan izuzetak u throw klauzuli, a može baciti izuzetak tipa IOException ili izuzetak IllegalDataException. Kada izuzetak izleti iz metode, program ulazi u normalan režim oporavka od greške, kao što je već opisano.

Slučaj upotrebe 2: **`void transferData(byte[] data) throw(IOException);`**

Ukoliko ova metoda/procedura baci izuzetak koji je deklarisala, sve je u redu i oporavak od greške teče na već opisan način. Međutim, kada ova metoda podigne izuzetak koji nije deklarisala (IllegalDataException), onda se događa poziv procedure std::unexpected. Podrazumevano ponašanje procedure std::unexpected je nasilno gašenje programa (terminate). Programer može da podmetne svoju proceduru za obradu ovakve neočekivane situacije korišćenjem funkcije std::set_unexpected (zadaje se pokazivač na funkciju koju procedure std::unexpected treba da pozove).

Slučaj upotrebe 3:**`void transferData(byte[] data) throw();`**

Ukoliko programer želi da nagovesti da metoda ne treba da baci nijedan izuzetak, može se navesti prazna throw lista. Ukoliko, u ovom slučaju, bude bačen bilo koji od dva pomenuta izuzetka, poziva se std::unexpected.

Umesto prazne liste, preporučuje se korišćenje novije notacije:

```
void transferData(byte[] data) noexcept;
```

Kad u try bloku neka metoda baci izuzetak (neposredno ili posredno), preskače se ostatak try bloka i nastavlja izvršavanje u catch grani koja prva odgovara po tipu bačenom izuzetku. Zbog toga catch grane treba tako organizovati da se prvo navedu grane za posebnije tipove izuzetaka, a opštije stavljati na kraj.

Postoji naopštiji tip catch grane koji prihvata sve izuzetke, a to je **`catch (...)`**. Ovakvu granu uvek treba staviti kao poslednju u nizu.

Ukoliko se u try bloku ne želi preskakanje ostatka bloka nakon metoda koja može da baci izuzetak, moguće je u tom slučaju ungezditi try blok u postojeći koji treba da obradi grešku osetljive metode. Primer:

```
try {
     try { method1(); } catch (exception& e1) { /*H1*/ }
     method2();
}
catch (SomeException& e) { /*H2*/}
```

U ovom slučaju, ukoliko method1 baci izuzetak, biće obrađen u grani H1 i posle toga se nastavlja sa izvršavanjem try bloka, u koje je sledeća naredba poziv metoda method2().

Standardna C++ biblioteka i izuzeci

Standardna C++ biblioteka u zaglavlju <exception> donosi klasu *exception* koja omogućava izvođenjem definisanje korisničnih tipova izuzetaka. VEOMA JE PREPORUČLJIVO dizajnirati izuzetke kao korisničke tipove (klase) izvedene iz exception, posredno ili neposredno. Nije preporučljivo bacati bilo koji tip koji nije izveden iz klase exception.

Isto tako, pojedine metode klasa iz standardne biblioteke, kao i neke operacije, bacaju takođe izuzetke kao objekte klase izvedenih iz klase exception. Primer nekih standardnih izuzetaka:

bad_alloc	Kad ne uspe alokacije operatorom new.
bad_cast	Kad ne uspe dynamic_cast.

Instanciranje izuzetaka, hvatanje, prosleđivanje i destrukcija

Kad je potrebno baciti objekat izuzetka napravi se automatski objekat izuzetka i baci naredbom throw. Na primer:

```
throw exception("something went wrong during...")
```

Objekat koji se ovom prilikom napravi smešta se u poseban bafer objekata za izuzetke kao bi preživeo razmotavanje steka. Objekat će prestati da živi nakon izvršavanje catch grane koja može da ga prihvati.

Kad se izuzetak hvata, preporučljivo je koristiti referencu na objekat u catch grani, kako bi se izbeglo nepotrebno kopiranje objekata, a i da se omogući supstitucija objekata, tako da naredna grana može da prihvati bilo koji objekat klase izvedene iz klase exception!

```
catch (exception& ex) { /* react on the error...*/ }
```

Ukoliko je određeni try-catch blok potrebno napisati kako bi se usled pojave izuzetka mogli osloboditi neki bitni lokalno zauzeti resursi, pri čemu i pored toga izuzetak treba da nastavi svoj put, pribegava se hvatanju i prosleđivanju izuzetka na sledeći način:

```
Resource res = nullptr;
try {
  res = new Resource(...);
  // the rest of the block.
catch (exception& ex) {
  /* release the resource here...*/
  if (res != nullptr) delete res;

  /*... and pass the exception.*/
  throw;
}
```

Uhvaćeni izuzetak se prosleđuje praznom throw naredbom.

8.2. Rad sa fajlovima (upotreba izuzetaka za obradu grešaka)

Implementirati na programskom jeziku C++ klase za rad sa fajlovima. Fajl ima logičku putanju na medijumu za čuvanje podataka, može da se obriše, da se proveri da li fajl postoji. Pisac (Writer) može da upiše tekstualni sadržaj u pridruženi fajl, može mu se pridružiti fajl za pisanje i može se završiti pisanje. Ukolio se pokuša pridruživanje novog fajla, iako stari još nije zatvoren, to je greška. Greška je i ako fajl ne može da se pronađe ili obriše. Ukoliko pisac ne može da upiše sadržaj u fajl, treba da prijavi grešku.

Rešenje:

```
#ifndef __EXCEPTIONS__H__
#define __EXCEPTIONS__H__

#include <exception>
#include <string>

using namespace std;

class FileException;
class FileNotFoundException;
class FileNotDeletedException;
class FileAlreadyAssignedException;
class FileAccessException;
class FileNotClosedException;

class File {
string filePath;
public:
   File (const string& path) : filePath(path) { }
   void deleteFile() throw (FileNotFoundException, FileNotDeletedException);
   bool exists() const;
   const string& getPath() const { return filePath; }
};

class Writer {
   File* assignedFile;
   ofstream* assignedStream;
public:
   Writer(): assignedFile(NULL), assignedStream(NULL) {}
   void assignFile(const File& file) throw (FileAlreadyAssignedException);
   void close() throw (FileNotClosedException);
   void write(const string& text) throw (FileAccessException);
   const File* getAssignedFile() const { return assignedFile; }
};


class FileException: public exception {
File* wrongFile;
public:
FileException(const string& msg, const File* file=NULL);
virtual ~FileException() { delete wrongFile; }
File* getWrongFile() const { return wrongFile; }
};

class FileNotFoundException: public FileException {
public:
FileNotFoundException(const string& msg, File* file=NULL)
   : FileException(msg, file) {}
};

class FileAccessException : public FileException {
public:
```

```
FileAccessException(const string& msg, File* file=NULL): FileException(msg, file) {}
};

class FileAlreadyAssignedException : public FileAccessException {
public:
FileAlreadyAssignedException(const string& msg, File* file=NULL)
   :FileAccessException(msg, file) {}
};

class FileNotDeletedException : public FileAccessException {
public:
FileNotDeletedException(const string& msg, File* file=NULL)
   :FileAccessException(msg, file) {}
};

class FileNotClosedException : public FileAccessException {
public:
FileNotClosedException(const string& msg, File* file=NULL)
   :FileAccessException(msg, file) {}
};

#endif


#include <iostream>
#include <fstream>
#include <stdio.h>
#include "exceptions.h"

using namespace std;

FileException::FileException(const string& msg, const File* file): exception(msg.c_str()) {
if (file != NULL)
   wrongFile = new File(*file);
else wrongFile = NULL;
}

bool File::exists() const {
ifstream infile(filePath);
return infile.good();
}

void File::deleteFile() throw (FileNotFoundException, FileNotDeletedException) {
if (!exists()) throw FileNotFoundException(string("File not found: ") + filePath, this);
if (remove(filePath.c_str()) != 0)
   throw FileNotDeletedException(string("File not deleted: ")+ filePath, this);
}

void Writer::close() throw (FileNotClosedException) {
assignedFile = NULL;
if (assignedStream)
   assignedStream->close();
assignedStream = NULL;
}
```

```
  void Writer::write(const string& text) throw (FileAccessException) {
  char* buffer = NULL;
  try {
    buffer = new char[text.size()+1]; // dinamicki alocirana memorija
    //... uradi se nesto sa znakovima u baferu nakon kopiranja..

    if (assignedStream == NULL || assignedStream->bad()) {
      string msg = (assignedFile != NULL) ? assignedFile->getPath() : "File not assigned";
      throw FileAccessException(string("Could not write to file: ") + msg, assignedFile);
    }
    *assignedStream << text << endl;

    delete [] buffer;  // na kraju se brise bafer, ali...
                // sta ako se pre toga pojavi izuzetak? Ova linija bice preskocena.
    buffer = NULL;
  }
  catch (FileAccessException& e) { // Zato, hvatamo izuzetak samo da bismo oslobodili
                                   // zauzete resurse.
    // Oslobadjaju se  resursi
    if (buffer != NULL) delete [] buffer;
    throw e; // Prosledjuje se izuzetak dalje.
  }
  }

  void    Writer::assignFile(const    File&    file)    throw    (FileAlreadyAssignedException,
FileNotFoundException){
  if (assignedFile != NULL)
    throw   FileAlreadyAssignedException(string("There   is   already   assigned   file:   ")   +
file.getPath(), &File(file));
  if (!file.exists())
    throw FileNotFoundException(
      string("Could assign file because, file not found:")+file.getPath(),
      &File(file));

  assignedFile = new File(file);
  assignedStream = new ofstream(assignedFile->getPath().c_str(), ios::app);
  }

  void safeDelete(File* f)  { // Sluzi za brisanje fajlova u granama za obradu izuzetaka,
                              // pa ne sme da baci nijedan izuzetak.
  try {
    if (f) f->deleteFile();
  }
  catch(FileException& e) {
    cerr << "Could not safely delete file: " << e.what() << endl;
  }
  }

  void safeClose(Writer& w)  { // Sluzi za zatvaranje u granama za obradu izuzetaka,
                               // pa ne sme da baci nijedan izuzetak.
  try {
    w.close();
  }
  catch(FileNotClosedException& e) {
    cerr << "Could not safely close file: " << w.getAssignedFile() << endl;
  }
  }
```

```
int main() {

Writer w;
File f (string("c:\\text.csv"));
try {
   w.assignFile(f);
   w.write(string("This is some text..."));
   // w.assignFile(f); // Ova linija koda sluzi za testiranje izuzetka.
   w.close();
}
catch (FileNotFoundException& e) {
   cerr << "ERROR: " << e.what() <<endl;
}
catch (FileAccessException& e) { // Na pocetku specijalniji izuzeci...
   cerr << "ERROR: " << e.what() << endl;
   safeClose(w);
}
catch (FileException& e) { // Najopstiji tip izuzetka na kraju..
   cerr << "ERROR: " << e.what() << endl;
   safeClose(w);
   safeDelete(e.getWrongFile());
}

return 0;
}
```

9. Šabloni (generički tipovi) i alokacija memorije

9.1. Šabloni funkcija.

Implementirati šablon funkcije Max koja pronalazi veću od dve zadate vrednonsti proizvoljnog tipa.

Rešenje:

```
/*
 * function_template.h
 *
 *  Created on: 24.12.2014.
 *      Author: Nemanja Kojic
 */

#ifndef FUNCTION_TEMPLATE_H_
#define FUNCTION_TEMPLATE_H_

template <typename T>
inline T const& Max (const T& a, const T& b)
{
    return a < b ? b:a;
}

#include <iostream>
using std::ostream;

class X
{
public:
   explicit X(int v) { val = v; }

private: // data members
   int val;

private: // friend functions
   friend bool operator<(const X& x1, const X& x2);
   friend ostream& operator<<(ostream& out, const X& x);
};


inline bool operator<(const X& x1, const X& x2)
{
   return x1.val < x2.val;
}

inline ostream& operator<<(ostream& out, const X& x)
{
   out << "x(" << x.val << ")";
}


#endif /* FUNCTION_TEMPLATE_H_ */
```

```cpp
/*
 * function_template.cpp
 *
 *  Created on: 24.12.2014.
 *      Author: Nemanja Kojic
 */

#include "function_template.h"

#include <iostream>
#include <string>

using std::cout;
using std::endl;
using std::string;
// using namespace std;


int main () {

    int i = 39;
    int j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;

    double f1 = 13.5;
    double f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;

    string s1 = "Hello";
    string s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;

    X x1(4);
    X x2(5);
    cout << "Max(x1, x2): " << Max(x1, x2) << endl;

   return 0;
}
```

9.2. Šabloni klasa

Implementirati generičku klasu za stek.

Rešenje:

```
/*
 * class_template.h
 *
 *  Created on: 24.12.2014.
 *      Author: Nemanja Kojic
 */

#ifndef CLASS_TEMPLATE_H_
#define CLASS_TEMPLATE_H_

#include <vector>

template <class T>
class Stack {
public:
   void push(const T&);
   void pop();
   T top() const;
   bool empty() const { return elems.empty(); }

private:
   std::vector<T> elems;     // elements
};

#endif /* CLASS_TEMPLATE_H_ */
```

```
/*
 * class_template.cpp
 *
 *  Created on: 24.12.2014.
 *      Author: Nemanja Kojic
 */
#include "class_template.h"

#include <stdexcept>

template <class T>
void Stack<T>::push (T const& elem)
{
   // Append copy of the passed element.
   elems.push_back(elem);
}

template <class T>
void Stack<T>::pop ()
{
   if (elems.empty())
      throw std::out_of_range("Stack<>::pop(): empty stack");
   // Remove the last element
   elems.pop_back();
}
```

```
template <class T>
T Stack<T>::top () const
{
if (elems.empty())
   throw std::out_of_range("Stack<>::top(): empty stack");
   // Return copy of the last element
   return elems.back();
}


// main.cpp
#include <iostream>
#include <stdexcept>  // standard exceptions
#include <string>

using std::string;
using std::cout;
using std::endl;

int main()
{
try
{
   Stack<int>      intStack;       // stack of ints
   Stack<string>   stringStack;    // stack of strings

   // manipulate int stack
   intStack.push(7);
   cout << intStack.top() << endl;

   // manipulate string stack
   stringStack.push("hello");
   cout << stringStack.top() << endl;
   stringStack.pop();
   stringStack.pop();
}
catch (std::exception const& ex)
{
   std::cerr << "Exception: " << ex.what() <<endl;
   return -1;
}
}
```

9.3. Alokacija memorije

Napisati program na jeziku C++ koji ilustruje upotrebu operatora new za bezbednu alokaciju memorije.

Rešenje:

```
/*
 * memory_allocation.cpp
 *
 *  Created on: 25.12.2014.
 *      Author: Nemanja Kojic
 */
// operator new example
#include <new>          // ::operator new
#include <cstdlib>
#include <stdexcept>
#include <iostream>     // std::cout

struct MyClass {
   int data[10000];  // large data member
   MyClass() {std::cout << "constructed [" << this << "]\n";}
};

const int CHUNK_SIZE = 100000;
const int MAX_CHUNKS = 5;

int next_chunk_idx = 0;

int* MEMORY_RESERVES[]
{
   new int[CHUNK_SIZE],
   new int[CHUNK_SIZE],
   new int[CHUNK_SIZE],
   new int[CHUNK_SIZE],
   new int[CHUNK_SIZE],
};

int activateReservedMemory()
{
if (next_chunk_idx >= MAX_CHUNKS)
   return -1;

// Free one chunk.
delete MEMORY_RESERVES[next_chunk_idx];
MEMORY_RESERVES[next_chunk_idx++] = nullptr;
std::cout
   << "*** Activated a chunk of the reserved memory (No."
   << next_chunk_idx << ")"
   << std::endl;

return 1;
}
```

```
void testStandardNew()
{
int iteration_cnt = 1;
while (true)
{
   try
   {
      std::cout << iteration_cnt << ":";
      MyClass* obj = new MyClass; // memory leakage - bad_alloc thrown!
      iteration_cnt++;
   }
   catch (const std::bad_alloc& ex)
   {
      // Log the error first.
      std::cout << "Ran out of memory (bad_alloc)!" << std::endl;
      // Fault tolerance: try to retrieve a memory from the reserves and continue.
      if (activateReservedMemory() != 1)
      {
         std::cout << "Ran out of memory - unable to recover!" << std::endl;
         // Finally, kill the program.
         exit(100);
      }
   }
}
}

void testNewWithNoThrow()
{
int iteration_cnt = 1;
while (true)
{
   std::cout << iteration_cnt << ":";
   // No exception. Test pointer instead.
   MyClass* obj = new (std::nothrow) MyClass; // memory leakage!
   if (obj == nullptr)
   {
      // Log the error first.
      std::cout << "Ran out of memory (NULL)!" << std::endl;
      if (activateReservedMemory() != 1)
      {
         std::cout << "Ran out of memory - unable to recover!" << std::endl;
         // Finally, kill the program.
         exit(100);
      }
   }
   iteration_cnt++;
}
}
```

```
int main()
{
   std::cout << "1: ";
   // Allocates memory by calling: operator new (sizeof(MyClass))
   // and then invokes the constructor of the newly allocated object.
   MyClass * p1 = new MyClass;

   std::cout << "2: ";
   // Allocates memory by calling: operator new (sizeof(MyClass), std::nothrow)
   // and then invokes the constructor of the newly allocated object.
   // Returns nullptr if there is no enough memory for allocation!
   MyClass * p2 = new (std::nothrow) MyClass;

   std::cout << "3: ";
   // Does not allocate memory -- calls: operator new (sizeof(MyClass), p2)
   // It just invokes the default constructor of the object.
   new (p2) MyClass;

   std::cout << "4: ";
   // Allocates memory by calling: operator new (sizeof(MyClass))
   // But does not call MyClass's constructor!
   MyClass * p3 = (MyClass*) ::operator new (sizeof(MyClass));

   delete p1;
   delete p2;
   delete p3;

   std::cout << "Enter \n1 for standard new, or \n2 for new without exception." << std::endl;
   int choice;
   std::cin >> choice;

   if (choice == 1)
      testStandardNew();
   else if (choice == 2)
      testNewWithNoThrow();
   else
      std::cout << "Wrong option selected" << std::endl;

   return 0;
}
```

10. Ispitni zadaci

10.1. Junski ispitni rok 2014/2015.

Bračne vode - praktični zadatak

Obavezni (eliminatorni) deo (15 poena).

***Datum* (*Date*)** je apstraktni tip podatka za predstavljanje validnih datuma, koji se sastoji od dana, meseca i godine. Stvara se zadavanjem dana, meseca i godine. Datum može da se ispiše u izlazni tok (operator<<), na datum se može dodati zadati broj dana (operator+), moguće je oduzeti datum od drugog datuma (operator-, rezultat je izražen u broju dana). Dva datuma je moguće uporediti na jednakost (operator==), kao i odrediti poredak dva datuma (operator> i operator<). Smatrati da godina traje 365 dana.

Kalendar (Calendar) omogućava dohvatanje trenutnog dana (*getToday(): Date*). Sme da postoji samo jedan kalendar u sistemu (implementirati Singleton DP). Moguće je postaviti datum kalendara (*setToday(date: Date)*).

Opis koncepata.

Osoba (Person) ima ime i prezime (*name*), datum rođenja (*birthDate:Date*) i pol (*gender*, M/F) koji mogu da se dohvate. Pravi se zadavanjem imena, datuma rođenja i pola. Može se proveriti da li je osoba punoletna (*isAdult():bool)* (uslov je da ima 18 godina i više), i može da se ispiše u izlazni tok (operator<<).

***Brak* (*Marriage*)** je zajednica dve osobe, od kojih je jedna suprug (*husband*), a druga supruga (*wife*). Pravi se zadavanjem supružnika, kao i datuma sklapanja braka. Brak može i da se prekine (*terminate(terminationDate:Date*)), pri čemu se zadaje datum prekida braka, koji predstavlja još jednu sastavnu informaciju braka (null ako je brak u trajanju). Moguće je proveriti da li je brak aktivan (*isActive():bool*, ako nema datum prekida).

U braku mogu nastati nove osobe koje se zovu deca. Osoba može da ima informaciju o braku iz koga potiče (ali i ne mora ukoliko taj podatak nije dostupan/poznat/relevantan). Jedna osoba može imati više brakova, od kojih je samo jedan (poslednji sklopljen) aktuelan, a takođe i ne mora biti u braku. Omogućiti dohvatanje trenutkog braka osobe (*getCurrentMarriage(): Marriage*), kao i dohvatanje braka u kom je osoba rođena (*getBirthMarriage(): Marriage*). Braku je moguće dodati dete (*addChild(d:Person)*).

Nekorektne situacije sklapanja/stvaranja braka treba prijaviti izuzecima tipa *InvalidMarriageException*. U ovom zadatku, brak nije validan ako su supružnici istog pola (*InvalidGenderException*), zatim ako bar jedan od supružnika nije punoletan (*JuvenileSpouseException*), ili ako je bar jedan od supružnika već u braku (*AlreadyMarriedException*).

Napisati glavni program koji treba da demonstrira funkcionalnosti na sledeći način:

Za 6 na praktičnom delu (15-19 poena).

Demonstrirati sve operacije sa klasama Calendar i Date.

Za 7 na praktičnom delu, dodatno uraditi sledeće (20-23 poena).

Napravi osobe sa sledećim podacima:

Petar Petrovic rođen 21.03.1984.
Nada Pavlovic rođena 3.5.1989.
Milan Petrović rođen 11.11.2013.
Jelena Markovic rođena 14.10.2011.
Marko Marković rođen 4.6.2000.
Mirka Mirković rođena 31.12.1988.
Dragan Marković rođen 4.4.1979.

Ispiše sve punoletne osobe. {Petar, Nada, Mirka, Dragan}.

Za 8 na praktičnom delu, dodatno uraditi sledeće (24-27 poena).

Demonstrirati sklapanje/raskidanje sledećih validnih brakova:

Petar i Nada, dana 10.04.2013.
Milan je dete rođeno u ovom braku.
Mirka i Dragan dana 23.11.2010.
Jelena je dete rođeno u ovom braku.
Brak između Mirke i Dragana se prekida 10.01.2015.

Mirka i Dragan više nemaju aktivan brak (raskinuti brak se svakako pamti kao deo istorije).
Mirka sklapa brak dana 8.3.2015. sa Milošem Pavlovićem rođenim 1.1.1990.
Iz ovog braka rađa se dete Strahinja Pavlović 23.06.2015. :)
Ispisati sve brakove koji traju više od godinu dana. Današnji dan je 22.11.2016.

Za 9 na praktičnom delu, dodatno uraditi sledeće (28-31 poena):
Za svaku maloletnu osobu ispisati osnovne podatke, kao i ime i datum rođenja za oba roditelja.
Za svaku osobu, koja je u braku, ispisati njenu decu (uključuje i decu iz prethodnih brakova).

Za 10 na praktičnom delu, dodatno ispitati sledeće nelegalne situacije: (32-35 poena):
Pokušaj sklapanja braka Mirke i Dragana, prekida se izuzetkom AlreadyMarriedException.
Pokušaj sklapanja braka Dragana i Vesne Zivkovic rođene 1.1.2001. prekida se izuzetkom JouvenilMarriageException.
Pokušaj sklapanja braka između Dragana i Marka prekida se izuzetkom InvalidGenderException.

Tehnička napomena:

Klase treba da uključuju, pored zahtevanih operacija/metoda, i sve druge metode koje obezbeđuju njihovo bezbedno korišćenje (konstruktori, destruktori, operatori). Takođe, gde je zgodno, a nije eksplicitno zahtevano, slobodno se mogu uvesti metode za dohvatanje/čitanje vrednosti atributa klasa uz strogo poštovanje principa enkapsulacije. Svi izuzeci treba da budu izvedeni iz jednog korenog izuzetka.