Materijali za vežbe iz Objektno orijentisanog programiranja

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++ (pritom, 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 allokaciju 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 upareno `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 parameter deklarisati kao: (1) pokazivač na prvi element niza (`int*`) ili (2) korišcenjem uglastih zagrada (koje implicitno označavaju niz, `int[]`). Na jeziku C++ moguce 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 ukljuce imena iz imenskog prostora direktivom `using namespace`, imenima je moguće pristupati navodenjem 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 "neogranicenoj tacnosti". 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 - Uredjivanje imena gradova smestenih u dinamicku 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 troglova 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:

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

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

c) `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 modifikatorom 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;
}
```

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

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 za Tačka 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 parameter 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. Funcija `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 modifikatorom 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 Tekst koja omogućava rad sa znakovnim nizovima.

Rešenje:

```
// File: tekst.h
// Klasa sa destruktorom.
class Tekst {
   char* niz;                          // Pokazivac na sam Tekst.
public:
   Tekst () {                          // Inicijalizacija praznog niza.
      niz = 0;
   }
   Tekst (const char*) ;               // Inicijalizacija nizom znakova.
   Tekst (const Tekst&) ;              // Inicijalizacija tipom Tekst.
   ~Tekst () ;                         // Unistavanje objekta tipa Tekst.
   void pisi () const;                 // Ispisivanje niza.
};


// File: tekst.cpp
// Implementacija metoda klase Tekst.
#include <cstring>
#include <iostream>
using namespace std;

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

Tekst::Tekst (const Tekst& t) {
   niz = new char [strlen(t.niz)+1];
   strcpy (niz, t.niz);
}

Tekst::~Tekst () {
   cout << "Destruktor[tekst -> \"" << ((niz!=NULL) ? niz : "NULL") << "\"]"<<endl;
   delete [] niz;
   niz = 0;
}

void Tekst::pisi () const {
   cout << niz;
}
```

```
void main () {
   Tekst pozdrav ("Pozdrav svima");   // Poziva se Tekst (char*).
   Tekst a = pozdrav;                 // Poziva se Tekst (Tekst&).
   Tekst b;                           // Poziva se Tekst ().
   cout << "pozdrav = ";
   pozdrav.pisi ();
   cout << endl;
   cout << "a       = ";
   a.pisi ()      ;
   cout << endl;
} // Ovde se poziva destruktor za sva tri objekta. Redosled destruktora?
```

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 resurs (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 redosledna njihovog nastanka.

2.7. Zadatak

Trgovačka firma (Company) prima narudžbine od proizvoljno mnogo klijenata (Customer). Svaki klijent šalje narudžbinu samo jednoj firmi, a narudžbina se sastoji iz 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 se ne pronađu artikli datog tipa, ispituje se da li postoje zamene za dati tip. Hijerarhija zamene između vrsta artikala uspostavlja se po sledećem principu: jedna vrsta može da zameni najviše jednu vrstu artikla, dok ona može biti zamenjena sa prozvoljno mnogo supstituta. Ukoliko u skladištu ne postoje ni artikli datog tipa, ni njegove zamene, kompanija šalje zahtev svom dobavljaču robe (Supplier).

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

Rešenje:

```
#ifndef __KONCEPTI__H__
#define __KONCEPTI__H__

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

class Item {
public:
   Item (const string& n): name(n) {}
   void addSubstituent(Item* it) { substituents.push_back(it); }

   vector<Item*> getSubstituents() const { return substituents; }
   Item* getSubstitutable() const { return substitutable; }
   const string& getName() const { return name; }

   bool checkIfSubstituent(const string& itemToCheck);
private:
   string name;
   Item *substitutable;
   vector<Item*> substituents;
};

class Customer;

class Demand {
public:
   Demand(Customer* c, const string& itName);
   const string& getItemName() const { return itemName; }
private:
   string itemName;
};

class Supplier {
public:
   void receiveDemand(Demand* d) {/* TODO: Implement this method! */}
};
```

```
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; }
private:
   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);
   Item* checkSubstituents(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; }
private:
   string name;
   vector<Demand*> myDemands;
   Company* myCompany;
};


#endif
```

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

bool Item::checkIfSubstituent(const string& itemToCheck) {
        if (substitutable) {
            if (substitutable->getName() == itemToCheck) return true;
            else return substitutable->checkIfSubstituent(itemToCheck);
        }
        return false;
}

void Company::receiveDemand(Customer* c, Demand *d) {
        Item* itemFound = myWarehouse->checkItem(d->getItemName());
        if (itemFound)
            c->notify(itemFound);
        else
            mySupplier->receiveDemand(d);
}

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

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

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

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

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

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

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

```
#include "trgovina.h"
#include <cstdlib>

int main() {

   Company* c = new Company("ACME");

   Customer* cust = new Customer("John Smith");
   cust->setCompany(c);

   cust->createDemand("DesktopPC");
   cust->createDemand("LaserPrinter");

   cust->sendOrder();

   return 0;
}
```

3. Rad sa ulaznim i izlaznim tokovima podataka

3.1. Zadatak

Napisati program na programskom jeziku C++ koji ilustruje rad sa ulaznim datotekam 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. Zadatak

Napisati na programskom jeziku C++ klasu komplesnih 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. Zadatak

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 Tekst {
  char* txt;                              // Pokazivac na sam tekst.
public:

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

  Tekst (const Tekst& tks) {              // Inicijalizacija Tkest-om.
    txt = new char [strlen(tks.txt)+1];
    strcpy (txt, tks.txt);
  }

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

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

int main () {
  Tekst a ("Dobar dan.");                 // Stvaranje i inicijalizacija.
  Tekst b = Tekst ("Zdravo.");
  Tekst 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 asmo kad levi i desni operand predstavljaju isti objekat i tada operatorska funkcija ne treba da radi ništa!

4.3. Zadatak

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); }
```

4.4. Zadatak

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. Zadatak

Ilustrovati preklapanje operatora() na primeru klase za predstavljanje instanci funkcije sinus. Svaka instanca se razlikuje prema parametrima funkcije. Omogućiti pozivom operatorske funkcije() 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. Zadatak

Ilustrovati preklapanje operatora ++ i -- na primeru klase časovnika Clock. Klasa Clock čuva podatke u satima i minutima, može da prmi jedan otkucaj, da ispiše vremen 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 clock forward by 1 min
   Clock operator++(int);       // postfix form; turn clock forward by 1 min

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

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;
};


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

void Clock::oneMinuteBack() {
   if (--min < 0) min = 59;
   if (min == 59) {
      if (--hour < 0) hour=23;
   }
}

void Clock::oneMinuteForward()
{
   min = (min + 1) % 60;
   if (min == 0)
      hour = (hour + 1) % 24;
}


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. Zadatak

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. Zadatak

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. Zadatak

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 sopstvenaTezina;
public:
  Vozilo (double st) { sopstvenaTezina = st; }
  virtual char vrsta () const =0;                    // Vrsta vozila.
  virtual double tezina () const { return sopstvenaTezina; }   // Ukupna tezina.
protected:
  virtual void pisi (ostream& it) const              // Pisanje.
    { it << vrsta() << '(' << sopstvenaTezina << ','; }
  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 PutnickoVozilo: public Vozilo {
  double srednjaTezinaPutnika;
  int brojPutnika;
public:
  PutnickoVozilo (double st, double srt, int bp): Vozilo (st)  // Konstruktor.
    { srednjaTezinaPutnika = srt; brojPutnika = bp; }
  char vrsta () const { return 'P'; }                  // Vrsta vozila.
  double tezina () const                               // Ukupna tezina.
  { return Vozilo::tezina () + srednjaTezinaPutnika * brojPutnika; }
private:
  void pisi (ostream& it) const                        // Pisanje.
    { Vozilo::pisi (it); it << srednjaTezinaPutnika << ',' << brojPutnika << ')'; }
};

#endif
```

```
// tvozilo.h - Klasa teretnih vozila.

#ifndef _tvozilo_h_
#define _tvozilo_h_

#include "vozilo.h"

class TeretnoVozilo: public Vozilo {
  double tezinaTereta;
public:
  TeretnoVozilo (double st, double t): Vozilo (st)          // Konstruktor.
    { tezinaTereta = t; }
  char vrsta () const { return 'T'; }                 // Vrsta vozila.
  double tezina () const                              // Ukupna tezina.
    { return Vozilo::tezina () + tezinaTereta; }
private:
  void pisi (ostream& it) const                       // Pisanje.
    { Vozilo::pisi (it); it << tezinaTereta << ')'; }
};

#endif

// vozilat.cpp - Ispitivanje klasa vozila.

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

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

  while (true) {
    cout << "\nVrsta vozila (T,P,*)? "; char vrsta; cin >> vrsta;
  if (vrsta == '*') break;
    switch (vrsta) {
      case 't': case 'T':
        cout << "Sopstvena tezina?     "; double sTez;  cin >> sTez;
        cout << "Tezina tereta?        "; double ter;   cin >> ter;
        vozila[n++] = new TeretnoVozilo (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;
        vozila[n++] = new PutnickoVozilo (sTez, srTez, brPut);
        break;
      default:
        cout << "*** Nepoznata vrsta vozila!\n";
    }
  }
  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;
  for (int i=0; i<n; delete vozila[i++]);
  return 0;
}
```

5.3. Zadatak

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`.

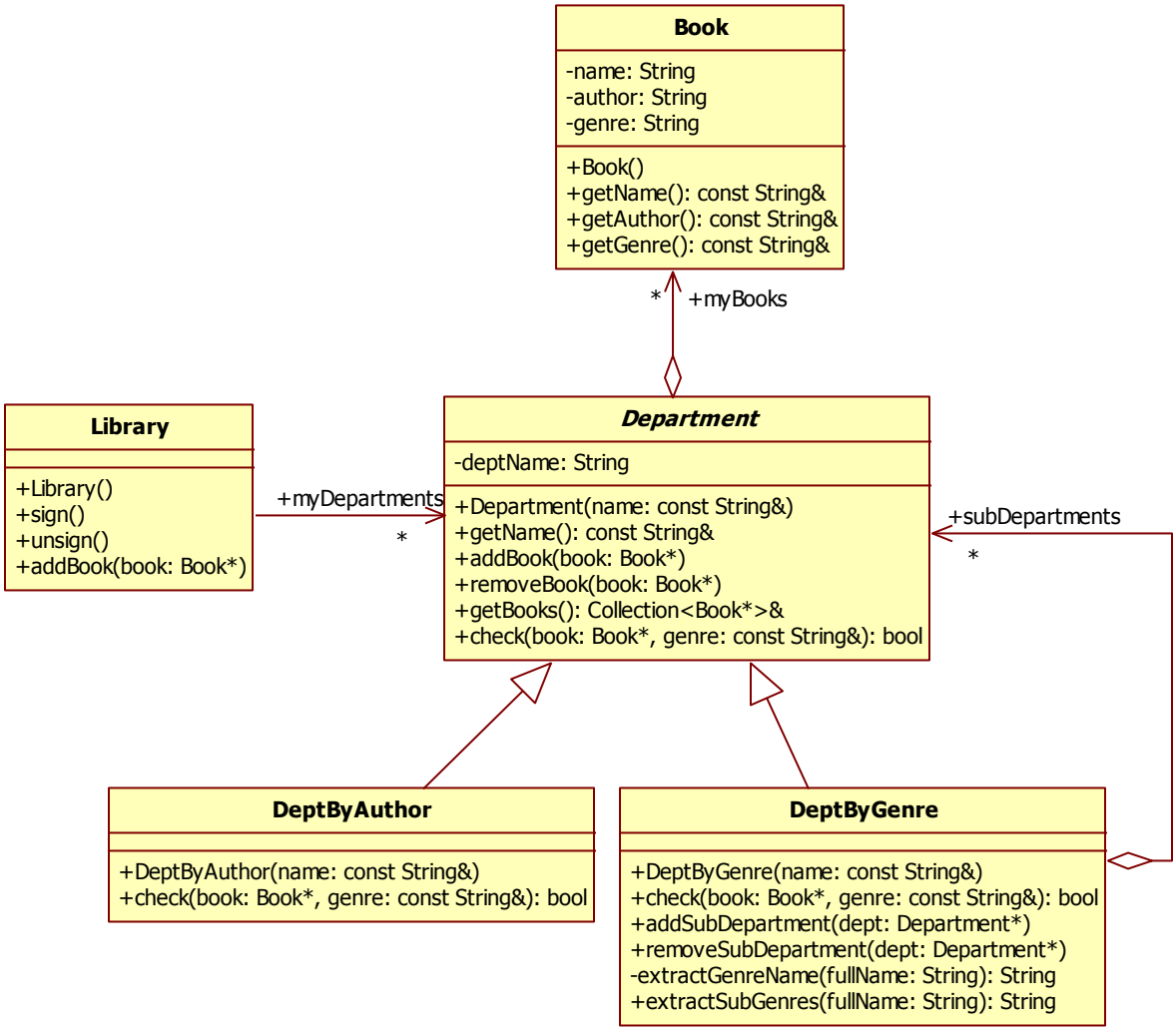
Napisati na programskom jeziku C++ klase za opisane koncepte. Napisati glavni program koji napravi biblioteku, dodaje joj odeljke i zatim testira stavljanje knjiga u biblioteku.

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;
};
```

6. Standardna biblioteka

6.1. Zadatak

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;
}
```

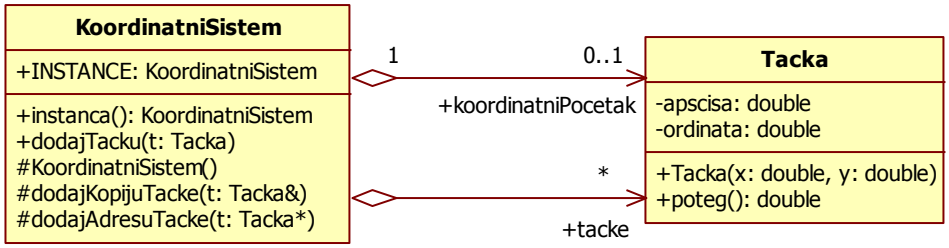
7. Projektni obrasci

7.1. Zadatak

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;
```

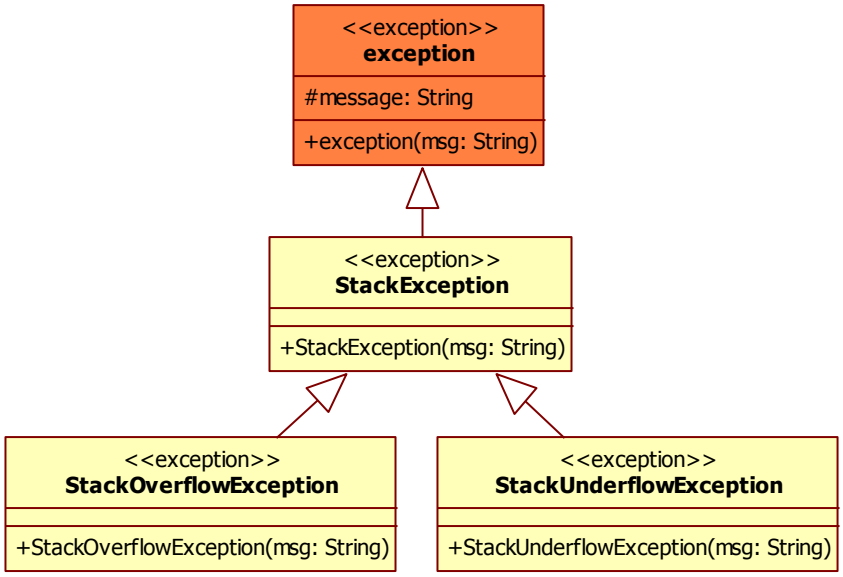
8. Obrada grešaka u programima – mehanizam izuzetaka

8.1. Zadatak

Napisati na programskom jeziku C++ program koji ilustruje obradu izuzetaka na primeru klase za stek.

Rešenje:

UML model izuzetaka




```
#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  size, curr;
   vector<int> content;
public:
   Stack(int capacity) :size(capacity), curr(0) {  }

   void push(int val) throw (StackOverflowException)
   {
      if (curr == size)
         throw StackOverflowException("Stack was full, when push was called!");
      content.push_back(val);
      curr++;
   }

   int pop() throw (StackUnderflowException)
   {
      if (curr == 0)
         throw StackUnderflowException("Stack was empty, when pop was called.");
      int result = content.at(curr-1);
      content.pop_back();
      curr--;
      return result;
   }
};

void testStack(Stack& s) throw (StackException)
{
   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 exception
   cout<<"POP =" << s.pop() << endl;
}

void main()
{
   Stack s(10);
   try {
      testStack(s);
      testStack(s);
      testStack(s);
   }
   catch(StackUnderflowException& se) {
      cout<<se.what() <<endl;
   }
   catch(StackException& se) {
      cout<<se.what() <<endl;
   }
   cout<<”END”<<endl;
}
```

8.2. Zadatak

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;
}
```