Osnovi računarskih sistema

Jelena Tomašević 2005/2006

Sadržaj

Čas 1

Beleške sa vežbi iz ORS-a (22.10.2005.)

1.1 Podsećanje sa prošlog časa

Reference, f-je (prenos argumenata, inline f-je,), složeni tipovi podataka (konstantni tipovi, nabrojivi, nizovi, strukture i unije), klase.
Nekoliko primera koji ilustruju korišćenje referenci:

Primer 1 *Ilustracija razlike u sintaksi pri korišćenju pokazivača i referenci.*

```
#include <iostream>
#include <string.h>

using namespace std;

struct tips
{
    int a;
    char s[1001];
};

/*
void cispis( const tips* x )
{
    cout << x->a << ' ' << x->s << endl;
}
*/

void ispis( const tips& x )
{
    cout << x.a << ' ' << x.s << endl;
}

/*
void cinit( tips* x, int a0, const char* s0 )
```

```
{
    x->a = a0;
    strcpy( x->s, s0 );
}
*/

void cinit( tips& x, int a0, const char* s0 )
{
    x.a = a0;
    strcpy( x.s, s0 );
}

main()
{
    tips a;

    for( int i=0; i<20; i++ ){
        init( a, i, "Zdravo!" );
        ispis( a );
    }

    return 0;
}
```

Primer 2 *Napisati funkciju koja vrši razmenu dva cela broja korišćenjem referenci.*

```
#include <iostream>

using namespace std;

void razmeni( int& x, int& y )
{
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;

    int t = x;
    x = y;
    y = t;

    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
}

main()
{
    int a = 5;
    int b = 10;
    cout << "a = " << a << endl;
```

```
    cout << "b = " << b << endl;

    razmeni( a, b );

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    return 0;
}
```

Rešimo sada jedan primer pomoću strukture, i isti taj primer pomoću klase i uporedimo ta dva rešenja:

1.2 Datum - rešenje pomoću strukture

Primer 3

```
#include <iostream>

using namespace std;

struct Datum
{
    int dan;
    int mesec;
    int godina;
};


void ispisDatuma( const Datum& d )
{
    cout << d.dan << '.' << d.mesec << '.' << d.godina << '.';
}


int prestupnaGodina( const Datum& d )//deljiva sa 4 i nije
     //deljiva sa 100 ili jeste deljiva sa 400
{
    return !(d.godina%4)
        && ( d.godina%100 || !(d.godina%400));
}

int brojDanaUMesecu( const Datum& d )
{
    switch(d.mesec){
        case 2:
            return
                prestupnaGodina( d )
                    ? 29
                    : 28;
        case 4:
```

```
        case 6:
        case 9:
        case 11:
            return 30;
        default:
            return 31;
        }
}

void sledeciDan( Datum& d )
{
    if( ++d.dan > brojDanaUMesecu(d) ){
        d.dan = 1;
        if( ++d.mesec > 12 ){
            d.mesec = 1;
            d.godina ++;
            }
        }
}


main()
{
    Datum d = { 28, 2, 2005 };
    ispisDatuma(d);
    cout << endl;
    sledeciDan(d);
    ispisDatuma(d);
    cout << endl;

    return 0;
}
```

1.3 Datum - rešenje pomoću klase

Da vidimo kako se ovaj zadatak može rešiti korišćenjem klase. Klasa se može shvatiti kao novi tip podataka - ima svoj naziv i operacije koje se mogu primenjivati na objekte(promenljive) tog tipa. S druge strane na klasu se može gledati i kao na nešto sasvim novo - nešto što je osnova drugačijeg koncepta programiranja (OOP-a).

Definicija klase:

```
class ime
{
//Deklaracija podataka clanova i f-ja clanica
};
Deklaracija se vrsi na sledeci nacin:
class ime;
```

Svaki od članova klase (podatak ili f-ja) može biti javni ili privatni (kasnije ćemo videti da postoji još jedna mogućnost!). Privatni(deklarisu se iza **private:**)su oni kojima se može pristupiti unutar klase a javni (deklarisu se iza **public:**) oni kojima se može pristupiti i izvan nje. Važenje svake od njih je do kraja bloka ili do pojave druge. Podrazumevano je private.
Preciznije definicija klase izgleda ovako:

```
class ime
{
public:
//Deklaracija javnih podataka clanova i f-ja //clanica.

private:
//Deklaracija privatnih podataka clanova i f-ja //clanica.

};
```

Pri tom redosled navodjenja delova nije od značaja.
Rešenje pomoću klase:

Primer 4

```
#include <iostream>

using namespace std;

class Datum
{
public:
    int dan;
    int mesec;
    int godina;

    void ispis()
    {
        cout    << dan << '.'
            << mesec << '.'
            << godina << '.';
    }

};

int prestupnaGodina( const Datum& d )
{
    return !(d.godina%4)
        && ( d.godina%100 || !(d.godina%400));
}

int brojDanaUMesecu( const Datum& d )
{
    switch(d.mesec){
        case 2:
```

```
            return
                prestupnaGodina( d )
                    ? 29
                    : 28;
        case 4:
        case 6:
        case 9:
        case 11:
            return 30;
        default:
            return 31;
        }
}

void sledeciDan( Datum& d )
{
    if( ++d.dan > brojDanaUMesecu(d) ){
        d.dan = 1;
        if( ++d.mesec > 12 ){
            d.mesec = 1;
            d.godina ++;
            }
        }
}


main()
{
    Datum d = { 28, 2, 2005 };
    d.ispis();
    cout << endl;
    sledeciDan(d);
    d.ispis();
    cout << endl;

    return 0;
}
```

F-je se mogu navoditi i u okviru same klase (uočiti razlike u broju argumenata!!!)

Primer 5

```
#include <iostream>

using namespace std;

class Datum
{
public:
    int dan;
    int mesec;
    int godina;
```

```
    void ispis() const
    {
        // const Datum* this;
        cout    << dan << '.'
            << mesec << '.'
            << godina << '.';
    }

    void sledeciDan()
    {
        // Datum* this;
        if( ++dan > brojDanaUMesecu() ){
            dan = 1;
            if( ++mesec > 12 ){
                mesec = 1;
                godina ++;
                }
            }
    }

    bool prestupnaGodina() const
    {
        // const Datum* this;
        return !(godina%4)
            && (godina%100 || !(godina%400));
    }

    int brojDanaUMesecu() const
    {
        // const Datum* this;
        switch(mesec){
            case 2:
                return
                    prestupnaGodina()
                        ? 29
                        : 28;
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            default:
                return 31;
            }
    }
};

main()
{
```

```
    Datum d = { 28, 2, 2005 };
    d.ispis();
    cout << endl;
    d.sledeciDan();
    d.ispis();
    cout << endl;

    return 0;
}
```

Dakle, sve što se može predstaviti pomoću strukture može i pomoću klase. Kakve nam to dodatne mogućnosti klasa pruža? Koncept klase nam omogućava da izvršimo skrivanje nekih informacija, da deo strukture objekta učinimo nevidljivim spolja.

Primer 6

```
#include <iostream>

using namespace std;

class Datum
{
public:
    void postavi_dan(int d)
    {
        _dan = d;
    }
    void postavi_mesec(int m)
    {
        _mesec = m;
    }
    void postavi_godinu(int g)
    {
        _godina = g;
    }
private:
    int _dan;
    int _mesec;
    int _godina;

}

main()
{
Datum d;
int d,m,g;
cout << " Unesi dan, mesec i godinu! " << endl;
cin >> d >> m >> g;
d.postavi_dan (d);
d.postavi_mesec (m);
d.postavi_godinu (g);
```

```
return 0;
}
```

Ne možemo napisati Datum.posavi_dan(d) jer je Datum zapravo tip a d je promenljiva tog tipa. Takođe ne možemo da napišemo d._dan zato što je _dan privatni podatak tog objekta. Da bi mogli da pristupimo ovim podacima potrebno je da definišemo metode kao npr. vrati_dan() , vrati_god()
Osim ovih metoda nedostaje još mnogo toga kao na primer:
Da se izvrši poređenje dva objekta:

```
Datum d1,d2;
 if (d1 == d2)
```

Mogućnost učitavanja i ispisivanja dva datuma:

```
cin >> d1;
cout << d1;
```

i tako dalje ali to se ostavlja studentima za vežbu! ;o)

Primer 7

```
#include <iostream>

using namespace std;

class Datum
{
public:
    void postavi_dan(int d)
    {
        _dan = d;
    }
    void postavi_mesec(int m)
    {
        _mesec = m;
    }
    void postavi_godinu(int g)
    {
        _godina = g;
    }
    void ispisi_dan()
    {
        cout << _dan << endl;
    }
    void ispisi_mesec()
    {
        cout << _mesec << endl;
    }
    void ispisi_godinu()
    {
        cout << _godina << endl;
    }
```

```
    void ispisi_datum()
    {
         cout << _dan << '.'
            << _mesec << '.'
            << _godina << '.' << endl;
    }


private:
    int _dan;
    int _mesec;
    int _godina;

}

main()
{
Datum d1,d2;
int d,m,g;
cout << " Unesi dan, mesec i godinu za prvi datum! " << endl;
cin >> d >> m >> g;
d1.postavi_dan (d);
d1.postavi_mesec (m);
d1.postavi_godinu (g);
cout << " Unesi dan, mesec i godinu za drugi datum! " << endl;
cin >> d >> m >> g;
d2.postavi_dan (d);
d2.postavi_mesec (m);
d2.postavi_godinu (g);

d1.vrati_datum();
d2.vrati_datum();

return 0;
}
```

1.3.1 Pokazivač this

Svaki objekat sadrži svoju kopiju podataka članova klase. Tako d1 ima svoje vrednosti za dan, mesec i godinu a d2 svoje. S druge strane postoji samo jedna kopija svake f-je članice. Dakle, i d1 i d2 pozivaju istu kopiju bilo koje od f-ja članica. Videli smo da f-ja članica koristi članove svoje klase bez korišćenja operatora za pristup članovima. Ako se metoda vrati_datum() pozove za objekat d1 onda ce podaci članovi (_dan, _mesec i _godina) kojima pristupa ova metoda biti podaci članovi tog objekta. Ako se pozove za d2 biće podaci članovi objekta d2. Postavlja se pitanje kako ova metoda zna kojim podacima da pristupi? Odgovor na to pitanje je pokazivač this. Svaka f-ja članica sadrži pokazivač na objekat za koji je pozvana. To je pokazivač this.
Ako imamo f-ju:

```
void ispisi_datum()
    {
        cout << _dan << '.'
            << _mesec << '.'
            << _godina << '.' << endl;
    }
```

nju kompajler prevodi u:

```
void ispisi_datum(Datum *this)
    {
        cout << this->_dan << '.'
            << this->_mesec << '.'
            << this->_godina << '.' << endl;
    }
```

a poziv f-je umesto

```
d1.vrati_dan();
```

postaje

```
vrati_dan (&d1);
```

Dakle svaki objekat ima kao član i implicitni pokazivač this na njega samog.On se ne navodi, on se podrazumeva. Ovaj pokazivač se može koristiti i eksplicitno u okviru f-je članice klase kao na primer umesto

```
void ispisi_datum()
    {
         cout << _dan << '.'
            << _mesec << '.'
            << _godina << '.' << endl;
    }
```

može:

```
void ispisi_datum()
    {
        // Datum* this;
        cout    << this->_dan << '.'
            << this->_mesec << '.'
            << this->_godina << '.';
    }
```

1.3.2 Tokovi

Da bi jedan program bio izvršen na računaru postoje dva načina. Kompilacija (prevodjenje celog programa pre izvršenja) ili interpretacija (prevodjenje jednog po jednog reda u toku izvršavanja). C++ spada u grupu jezika koji se kompajliraju. U tom postupku program (koji se zove još i izvorni program - source) prolazi kroz tri osnovne faze: Predprocesiranje, procesiranje i povezivanje. Predprocesor priprema datoteku za prevodjenje. Naredbe koje se upucuju njemu

(zovu se i direktive pretprocesora) počinju znakom #. Njima se (izmedju ostalog) može zahtevati umetanje druge datoteke u tekst programa. Za to služi direktiva #include <naziv>, gde je naziv ime datoteke. Bitno je znati da li je nešto ugradjeno u jezik (operatori i naredbe)ili je definisano u nekoj drugoj datoteci (matematičke f-je,) i u kojoj je datoteci definisano. Nakon pretprocesiranja dobijeni rezultati idu u fazu procesiranja(kompajliranja) iz koje se dobijaju objektne datoteke sa ekstenzijom .obj. Zatim sledi povezivanje (linkovanje) čiji rezultat je izvršna datoteka sa ekstenzijom .exe - executable. Samo izvršne datoteke se mogu izvršavati. Pretprocesorske direktive su i npr. #define, #ifdef,#ifndef

Ulazno/izlazna funkcionalnost nije ugradjena u jezik c++ vec je ona obezbedjena kao deo standardne c++ biblioteke. Biblioteka koja pruza u/i funkcionalnost naziva se Biblioteka ulaznih i izlaznih tokova ili na englesom iostream biblioteka. U ovoj iostream biblioteci, datoteke se interpretiraju kao sekvence ili tokovi bajtova. Ulazne operacije su ugradjene u klasu istream (input stream, ulazni tok) a izlazne u klasu ostream (output stream, izlazni tok). Klasa iostream nasledjuje obe ove klase i omogućava dvosmernu komunikaciju. Dva predefinisana toka su:

1. cin, objekat klase istream vezan za standardni ulaz.
2. cout, objekat klase ostream vezan za standardni izlaz.

Za izlaz se koristi operator prosledjivanja <<. To je binarni operator koji se upotrebljava u infiksnoj notaciji: Levi operand je tok kome se prosledjuju podaci a desni operand je objekat koji se prosledjuje. Rezultat je referenca na izlazni tok (objekat klase ostream), čime je omogućeno nadovezivanje više primena ovog operatora.
cout $<<$ *x* $<<$ *y*;

Za ulaz se koristi operator izdvajanja >>. To je binarni operator koji se upotrebljava u infiksnoj notaciji. Levi operand je tok iz koga se izdvajaju podaci a desni operand je objekat čiji se sadrzaj izdvaja iz toka. Rezultat je referenca na ulazni tok (objekat klase istream), čime je omogućeno nadovezivanje više primena ovog operatora.
cin $>>$ *x* $>>$ *y*;

Primer 8 *Ilustracija operatora izdvajanja >> i prosledjivanja <<.*

```
#include <iostream>
#include <string>

using namespace std;

main()
{
/*
    cout << "Zdravo!\n";
    cout << 1 << 2 << 'a' << "niska" << endl;

    int x, y;
    cout << "Upisite dva broja: ";
    cin >> x >> y;
    cout << (x+y) << endl;
```

```
    char niska[100];
    cout << "Upisite nisku: ";
    cin >> niska;
    cout << niska << endl;

    char c,d,e;
    cout << "Upisite tri znaka: ";
    cin >> c >> d >> e;
    cout << c << d << e << endl;
*/
    string s; // ovo je objekat s klase string
    cout << "Upisite nisku: ";
    cin >> s;
    cout << s << endl;

    const char* cs = s.c_str();
    cout << cs << endl;
    cout << s.length() << endl;
    cout << (s+s) << endl;
    cout << s[2] << endl;

    const int ca = 17;
    cout << ca << endl;




    return 0;
}
```