Programabilni uređaji i objektno orjentisano programiranje

Preklapanje operatora

Preklapanje operatora pridruživanja

- Posmatrajmo sljedeći primjer **neupotrebljive** klase sa podatkom članom koji je pokazivač:

```
class X{
private:
      int *p;
public:
      X(){p=0;}
      X(int i):p(new int(i)){}
      X(const X&x):p(new int(*x.p)){}
      ~X() {delete p; p=0;}
      X &operator=(const X&);
      //...Još koješta
   };
```

```
X& X::operator=(const X& x){
      if(this != &x) {
      delete p;
      p = new int(*x.p);
}
      return *this;}
```

Što znači **`this != &x`**?

Tumačenje preklapanja kod pridruživanja

- Prethodna realizacija vodi računa o nekoliko stvari.
- Što treba uraditi ako neko napiše:
  `A = B;`
  za objekte klasnog tipa X koji sadrži pokazivač?
- Objekat `A` kojem želimo pridružiti vrijednost objekta `B` je već postojao i zauzimao neku memoriju za svoje podatke članove!
- Prvo treba uništiti prethodni sadržaj objekta A za koji se ova funkcija poziva (**`A = B`** je ekvivalentno **`A.operator=(B)`**). Ovo je važno učiniti da prethodni sadržaj ne bi nastavio da živi u memoriji nepotrebno je popunjavajući i potencijalno uzrokujući greške. Naročito je važno ukoliko imamo pokazivač na nizove, gdje se može desiti da niz na koji ukazuje pokazivač objekta **`B`** ima manje ili više elemenata od niza na koji je ukazivao pokazivač objekta **`A`**. Dakle, nije nam uvijek potreban isti kapacitet memorije za novu vrijednost koju dobija objekat **`A`** i za onu koju je već imao.

Tumačenje preklapanja pridruživanja

- Zatim, ne postavljamo prosto `p=x.p;` jer bismo na taj način izjednačili pokazivače i oni bi pokazivali na istu memorijsku lokaciju, odnosno A i B ne bi bili nezavisni objekti.
- Stoga smo, zapravo, formirali novi pokazivač koji pokazuje na objekat (u našem primjeru cijeli broj) koji je identičan onom na koji pokazuje pokazivač iz objekta argumenta X. Međutim, dva objekta na koje pokazivači pokazuju nalaze se na različitim memorijskim adresama, odnosno objekti A i B su nezavisni.
- Preostaje pitanje zbog čega ove operacije vršimo ako je `this!=&x`?

  Odgovor je da bismo izbjegli situaciju `A = A;` jer bi tada naredba `delete` izbrisala objekat za koji želimo izvršiti dalje operacije.
- Zbog čega vraćamo rezultat: `return *this;`?

  Zbog eventualne konkatenacije pridruživanja `A = B = C;`.

Operator konverzije

- Već smo vidjeli da se konverzija iz standardnog tipa podataka u klasni tip može vršiti preko konstruktora.
- Taj konstruktor kao argument uzima dati tip podatka i na osnovu njega formira klasni tip.
- Na isti način se može izvršiti konverzija iz jednog u drugi klasni tip podatka: preko konstruktora koji ima kao argument podatak klasnog tipa.
- Postavlja se pitanje kako se može izvršiti konverzija iz klasnog tipa u standardni tip.
- Očigledno mi nemamo pristup “konstruktoru” za ugrađene (standardne) tipove, jer se oni realizuju hardverski.

Preklapanje operatora konverzije

- Problem konverzije klasnih u standardne tipove podataka prevazilazi se preklapanjem operatora konverzije. To je specijalna operatorska funkcija koja nosi ime tipa u koji se konverzija želi izvršiti. Npr. za klasu complex:

```
operator double(){return real;}
```

- Malo iznenađenje je da ovdje ne navodimo tip rezultata koji funkcija vraća, ali se to podrazumijeva na osnovu naziva operatorske funkcije. Mogući načini pozivanja:

```
complex c(2,3.4);
double d1 = c;
//implicitna konverzija
d1 = 5 + double(c); d1 = 3 * (double)c;
//eksplicitna konverzija
```

Operator ()

- `operator()` je jedna od specijalnih operatorskih funkcija za koju važe specifična pravila.
- Ovaj operator mora biti nestatička funkcija članica, ali operatorska funkcija može da ima proizvoljan broj argumenata.
- Ako se sjećamo, ovaj operator se pored davanja prioriteta operacija može koristiti i za poziv funkcije, a funkcije mogu da imaju proizvoljan broj argumenata.
- Standardna primjena ovog operatora u C++ je obično "pametno indeksiranje" ("smart indexing").

operator () – smart indexing

- Npr. pretpostavimo da imamo klasu `S` koja predstavlja neki tekst. Tada, ako imamo neki objekat ove klase `S s;`, indeksiranje `s(a,b);` gdje je `a` recimo neki string, a `b` neki cijeli broj, možda može da vrati kao rezultat recimo pokazivač na poziciju u tekstu `s` na kojoj se po `b`–ti put pojavljuje string `a`.
- Posmatrajmo jedan jednostavan primjer. Neka posjedujemo klasu koja realizuje apstrakciju matrice. Za matricu su nam obično potrebni pokazivač na podatak određenog tipa (obično pokazivač na pokazivač) i dva cijela broja koji predstavljaju dimenzije matrice.

operator () kod klase matrica

- Dajmo samo ilustrativni dio ove klase:

```
class Matrica{
private:
    T **m; int r,k; //pretpostavka je da je
    //klasni tip T, ali to ne mora da bude
public: //konstruktori i ostale funkcije
    T &operator()(int, int);};
T & Matrica::operator()(int i,int j){
    if(i>0 && i<=r && j>0 && j<=k)
        return m[i-1][j-1];
else //neka poruka o grešci
}
```

- Predmetna funkcija vrši indeksiranje u granicama od 1 pa na dalje (za razliku od C++ i C koji to rade od nule) i ujedno informiše o eventualnim prekoračenjima.

Primjer poziva funkcije

- Jedan neobičan način da se ova funkcija pozove je:

```
Matrica P; //slijedi niz naredbi kojim se
//inicijalizuje matrica
P(2,2) = 4;
```

- Ilustrujemo praktično zbog posljednjeg reda, koji je, premda izgleda krajnje logično, ujedno i krajnje neobičan.
- Naime, ovo što piše nije MATLAB već predstavlja poziv funkcije: `P.operator()(2,2) = 4;` i radimo ono što smo učili da je nemoguće, odnosno rezultat operacije pridružujemo funkciji, a ne promjenljivoj na lijevoj strani.
- Međutim, rezultat ove funkcije je referenca, a to znači i promjenljiva u punom svom kvalitetu koja je alocirana u memoriji (i dostupna funkciji preko pokazivača this), pa je moguć ovaj izuzetno elegantan zapis!

```cpp
#include <iostream>
using namespace std;
class Matrica{
    private:
        int **m;int r,k;
         //pretpostavka je da je klasni tip T, ali to ne mora da bude
    public: //konstruktori i ostale funkcije
        Matrica(){m = 0;}
        Matrica(int,int,int **);
        int &operator()(int, int);
        };
Matrica::Matrica(int a,int b,int **c):r(a),k(b){
    m = new int *[a*b];  //zauzmemo u komadu memoriju za sve podatke
    for(int i = 0; i < r; i++){
        m[i] = m[0] + i * k;
        for(int j=0; j<b; j++)
            m[i][j] = c[i][j];}
    }

int & Matrica::operator()(int i,int j){
        if(i>0 && i<=r && j>0 && j<=k)
            return m[i-1][j-1];
        else
            cout<<"indeks prevazilazi dimenzije matrice"<<endl;
        }
```

```
int main()
{
    int cc[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    int **mm;
    mm = new int*[3];
     for(int i = 0; i < 3; i++){
        mm[i] = new int[4];
         for(int j = 0; j < 4; j++)
            mm[i][j] = cc[i][j];

    }
Matrica m1(3,4,mm);
    for(int i = 1; i <= 3; i++){
        for(int j = 1;j <= 4;j++)
            cout<<m1(i,j)<<" ";
        cout<<endl;}
    m1(2,3) = 16;
    cout<<m1(2,3);
    return 0;
}
```

Druge primjene operatora ()

- Pored navedene primjene, često se ovakav operator koristi kod lista za indeksiranje, recimo, elementa liste koja sadrži dati cijeli broj kao ključ.
- Pored ovoga, postoje i varijante kada se operator () koristi da bi specificirao koja će funkcija za neki objekat biti izvršena za neku operaciju. Tada je jedan od argumenata funkcije pokazivač na funkciju članicu klase (mada može i za nečlanicu).
- Više detalja o ovim i drugim primjenama pročitajte u knjizi od Milićeva (strane 284-286).

operator []

- I ova preklopljena operatorska funkcija mora biti članica (nestatička), ali fiksno sa jednim argumentom.
- Naime, operacija `x[y]` se smatra binarnom (učestvuju dva operanda) `x` i `y`. Pošto je u pitanju funkcija članica onda ona mora imati fiksno jedan argument, pošto je drugi argument objekat za koji se funkcija poziva (u ovom slučaju `x`).
- Primjena ovog operatora je uobičajeno (ali ne i ograničeno) vezana za pametno indeksiranje. Detalji, knjiga strane 286-287.

operator ->

- Ponovo je u pitanju (nestatička) funkcija članica koja mora biti bez argumenata!?
- Dakle, ovaj operator se smatra unarnim!
- Ako je poziv oblika `x -> y,` postavlja se pitanje koja se funkcija i za koji objekat poziva.
- Funkcija se poziva za objekat `x` i ovo zapravo odgovara pozivu `(x.operator->()) -> y`.
- Postavlja se onda pitanje što ova funkcija mora da vrati kao rezultat.
- Po analogiji sa strukturama ova funkcija mora da vrati pokazivač na objekat klasnog tipa koji ima podatak član `y`.

Primjena operatora ->

- Ova operatorska funkcija je najmaglovitija od svih.
- Pored pametnog indeksiranja, ova funkcija se koristi i u kombinaciji sa drugim operatorima (npr. operatorom dodjele), da bi produkovala neke dodatne efekte vezane za pokazivače. Npr. funkcija koja broji koliko se puta pristupalo putem pokazivača pojedinim objektima.
- Neke šture detalje možete pročitati u knjizi na strani 288.

Operatori new i delete

- Cilj operatora **`new`** je da alocira memoriju za objekat.
- On se, dakle, poziva kada objekat ne postoji u memoriji.
- Kod klasa se jedino statičke funkcije mogu pozivati kada objekat ne postoji (pozivaju se za klasu), pa stoga operator new mora biti statička funkcija članica!!!
- Cilj operatora **`delete`** je da izbriše objekat iz memorije. Odnosno, na kraju njegovog djelovanja ne postoji u memoriji objekat za koji je pozvan. Jedina funkcija koja nam ovu funkcionalnost može odraditi je opet statička funkcija članica.
- Pored ovoga, operatorske funkcije **`new`** i **`delete`** zadovoljavaju niz drugih pravila.

Operator new - pravila

- `operator new` mora da vrati pokazivač na `void`! da bi bio kompatibilan sa ostalim funkcijama koje vrše alokaciju memorije. Ovaj pokazivač je zapravo pokazivač na zauzeti memorijski prostor.
- Prvi argument ove funkcije je specijalnog tipa (`size_t`) koji označava kolika je zona u memoriji zauzeta, a pored ovog mogu postojati i drugi argumenti.
- `operator new` mora zauzimati memoriju, a ne smije pozivati eksplicitno konstruktor, jer će konstruktor biti implicitno "odrađen".
- `operator new` se izvršava prije konstruktora (zauzima memoriju).

Operator new i delete - pravila

- Prilikom poziva `operator new` funkcije često se koristi poziv ugrađenog operatora, a to se postiže sa `::operator new (arg)` (gdje je `arg` argument za koji se operator poziva).
- `operator delete` mora da ima prvi argument koji je tipa `void *` (pokazuje na zonu koja se dealocira). Rezultat je uvijek `void`. Funkcija može imati i druge argumente.
- `operator delete` ne smije da pozove eksplicitno destruktor (to se radi implicitno), već on treba samo da oslobodi memorijski prostor (stoga ga kompajler izvršava nakon destruktora).

operator delete - pravila

- U preklopljenoj operatorskoj funkciji `delete` često se poziva ugrađeni operator sa `::operator delete(p)`, gdje je `p` pokazivač na memoriju koja se želi dealocirati.
- Operatorske funkcije `new` i `operator delete` se ne mogu dodatno preklapati!
- Grubo govoreći, `operator new` i `operator delete` treba da rade isto što i standardni operatori, uz možda još neku malu dodatnu fleksibilnost i ništa više.
- Ako zatreba dealokacija nizova mora se preklopiti operatorska funkcija `operator delete []`, jer je to poseban operator.
- Preklopljeni operator `delete` može da sakrije činjenicu da vrši dealokaciju većeg broja klasnih objekata (obično liste ili sličnog elementa).

Preklapanje operatora - zaključak

- Neke detalje vezane za preklapanje operatora **`new`** i **`delete`** možete pročitati u knjizi na stranama 289-292.
- Jedini operatori koje detaljnije nijesmo objasnili su **`<<`** i **`>>`**.
- Ovo su binarni operatori i njihovo standardno preklapanje se ne razlikuje od preklapanja drugih binarnih operatora.
- Ovi operatori se, međutim, mogu koristiti za preklapanje kod formatizovanog ulaza i izlaza, ali to ćemo učiti tek nešto kasnije.

Primjer

- Iz knjige od Milićeva preuzet je dosta kompleksan primjer klase sa kompleksnim brojevima (strane 388-390).
- Pokušali smo da program bude u skladu sa našim kompajlerom, te da bude ispravan od grešaka, ali uvijek postoji mogućnost da nam se potkrade neka greška.
- Glavni program `main_complex.cpp` služi samo za testiranje.
- Jedina dva detalja koja vrijedi objasniti vezana su za zaglavlje u kojem uključujemo fajl sa realizacijom malo složenijih funkcija `complex.cpp` i iostream.h biblioteku uz jednu napomenu koju ćemo kasnije saopštiti.

Primjer – complex.h

- Definicija klase sa dvije promjenljive članice i određenim brojem funkcija, što prijatelja, što članica, nalazi se u header fajlu `complex.h`.
- U biblioteci, između ostalog, treba uočiti način na koji je spriječeno višestruko uključivanje ove biblioteke u okviru drugih programa.
- Ovaj fajl koristi `iostream.h` biblioteku koju koristi i glavni program, što sugeriše da je ova biblioteka zajedno sa funkcijama dva puta uvedena u program što nije dozvoljeno.
- Međutim, naš program radi i pored toga, što znači da postoji mehanizam u `iostream.h` biblioteci koji sprečava višestruku definiciju elemenata iz biblioteke `iostream.h`.
- Obično sve standardne biblioteke ovo poštuju.

Realizacija funkcija

- Od važnijih detalja vidimo da funkcija `operator *=` koristi množenje kompleksnih brojeva koje je realizovano u funkciji `operator *`.
- Na ovaj način se štedi u kodiranju, odnosno smanjuje prostor za grešku, i što je najvažnije obezbjeđuje činjenica da funkcije sa sličnom idejom, kao što je recimo `*` i `*=`, daju kompatibilno ponašanje.
- Sve jednostavne funkcije, bilo članice, bilo prijatelji, su realizovane u `complex.h` fajlu, dok su nešto složenije funkcije realizovane u `complex.cpp` fajlu.
- Fajl `complex.cpp` se može prevesti ako je sintaksno ispravan i u mašinski kod koji se ne može izvršavati (fajl sa ekstenzijom obj).

Primjer – complex.cpp

- `obj` verzije su mašinski kod u relativnim adresama koji se ne može izvršavati, ali se prevedena verzija može davati korisnicima na povezivanje sa drugim modulima, a bez uvida u izvorni (tekstualni) kod.
- Prevođenje do OBJ bez grešaka sugeriše da je dio programa sintaksno ispravan, ali ne može da garantuje da kod radi ono što smo mi očekivali da radi.
- U fajlu `complex.cpp` jedna funkcija je relativno složena, a riječ je o funkciji `operator/`. Mi smo se poslužili činjenicom da se rezultat djeljenja `z1/z2` ne mijenja ako se imenilac i brojilac pomnože sa `z2*`, a u imeniocu se dobija $|z2|^2$.

Primjer – Završni komentari

- Konačno, napomenimo samo još i činjenicu da nijesmo koristili, mada smo ih definisali, funkcije za štampanje i učitavanje kompleksnih brojeva, ali ćemo to raditi dosta kasnije kada se upoznamo sa ostalim OO konceptima.

Rekapitulacija

- Do sada smo izučavali:
  - Elemente C++ koji nijesu OO prirode, a koji su donekle izmjenjeni u odnosu na C;
  - Klasu i koncepte koji su sa klasom asocirani – enkapsulaciju i apstrakciju;
  - Prijatelje;
  - Operatorske funkcije.
- Ovo je približno najvažnija “trećina” gradiva vezanog za OO sintaksu.
- Spremajte se za II kolokvijum!!!

Ostatak gradiva

- Ostatak gradiva je podijeljen grubo u dva dijela:
  - Drugi dio čine:
    - Nasljeđivanje
    - Šabloni
    - Ova dva elementa približno zauzimaju nešto više od tri predavanja.
  - Treći dio se sastoji od:
    - Formatizovani ulaz/izlaz u C++ (nekoliko puta do sada pomenut)
    - Obrada izuzetaka
    - Ako ostane malo vremena biće nekoliko riječi o STL-u.