Napredne tehnike programiranja

Vježba 6

Teme

- Konstruktor i destruktor
- Defaultni konstruktor
- Konstruktor kopije

Konstruktor i destruktor

- Konstruktor i destrukor su posebne funkcije koje možemo dodati strukturi/klasi i koje se izvršavaju automatski svaki put kad se kreira/nestane objekt koji pripada toj strukturi/klasi

Konstruktori

- Konstruktor je specijalna funkcija koja ima isto ime kao i sama klasa

```
class stack {
      int vrh, podaci[100];
      public:
            stack( void ){
                  vrh = 100;
            }
};
```

- Konstruktoru se nikad ne piše povratni tip!!! (iako je sama ta funkcija uvijek void)

Konstruktori

- Alternativni zapis (kad je tijelo funkcije van strukture)

```
struct stack {
        int vrh, podaci[100];
        public:
                stack();
                ...
};

stack::stack(){
        vrh = 100;
}
```

Konstruktori

- Konstruktor se sam, automatski pozove čim deklariramo varijablu (objekt)!

```
int main( void )
{
      stack S, T; // pozovu se konstruktori
      cout << S.vrh; // ispise 100
      cout << T.vrh; // ispise 100
}
```

Destruktor

- Destruktor je posebna funkcija (član klase) koja se automatski poziva kad objekt koji pripada toj klasi prestaje postojati
- Objekt nestaje kad iziđe iz dosega (scope), kad program završi sa radom ili kad se za njega izvrši naredba delete
- Destruktor nema povratni tip niti argumente
- Klasa može imati samo jedan destruktor
- Destruktor ima isto ime kao klasa sa oznakom ~

Destruktori

- Često treba nešto napraviti kada objekt završava životni vijek (npr. osloboditi memoriju)
- tome služe destruktori isto nemaju povratnu vrijednost i automatski se pozivaju

```
class stack {
        int vrh, podaci[100];
        ~stack()
        {
                cout << "Stack je gotov!";
        }
};
```

Zadatak

- Što ispisuje ovaj program?

```
class test {
      test( void )
      { cout << "Rodio se novi test!"; }
      ~test( void )
      { cout << "Nestao je jedan test!"; }
};

int main( void ){
      test A, B;
      cout << "nesto";
      return 0;
}
```

Konstruktori

- Klasa može imati više konstruktora (overloading)
- Konstruktori se moraju razlikovati po broju ili tipu argumenata
- Ako ne postoji niti jedan konstrukor u klasi prevoditelj automatski kreira defaultni konstruktor bez argumenata (on kreira prazan objekt)

Konstruktori sa parametrima

- Konstruktor može imati i parametre

```
class stack {
      int vrh, *podaci;
      stack( int velicina ){
            podaci = new int [100];
            vrh = velicina-1;
      }
      ~stack() { delete [] podaci; }
};
```

Konstruktori sa parametrima

- u tom slučaju i objekte treba deklarirati sa odgovarajućim parametrima koji se onda prosljeđuju konstruktoru

```
class stack {
        int vrh, *podaci;
        stack( int velicina )
        {
                podaci = new int [100];
                vrh = velicina-1;
        }
        ~stack() { delete [] podaci; }
};

int main( void )
{
        stack S( 50 ); // stog sa 50 elemenata
        stack T( 100 ); // stog sa 100 elemenata
}
```

Konstruktor i destruktor

- Konstruktor i destruktor možemo koristiti i za dinamičku alokaciju memorije

```
#include <iostream>
using namespace std;
 class test {
        int *a;
        public:
        test(  ){
          a=new int;
          *a=0; }
        test (int b){
         a=new int;
         *a=b;
        }
        ~test( void )
        {
           delete a;
        }
    void ispis(){
            cout<<*a<<endl;
    }
};

int main ()
{
   test A, B(5);
   cout << "nesto"<<endl;;
   A.ispis();
   B.ispis();
   system ("pause");
   return 0;
}
```

Defaultni konstruktor

- Defaultni konstruktor je konstruktor bez argumenata
- On se automatski poziva svaki put kad kreiramo novi objekt (bez argumenata)
- Ako u strukturi ne kreiramo odgovarajući defaultni konstruktor kompajler ga kreira prilikom prevođenja

Defaultni konstruktor

- Program ne javlja grešku

```
class B{
      int x;
      /* struktura nema default
      konstruktor
}

int main(){
      B b; /* poziva se konstruktor koji
      kreira kompajler B(){}
}
```

Defaultni konstruktor

- Ako struktura ima konstruktor sa argumentima, a nema defaultni onda kompajler ne kreira defaultni konstruktor
- Ako se kreira novi objekt bez parametara dolazi do greške jer se ne može pozvati defaultni konstrukor (jer ne postoji)

Defaultni konstruktor

- Program javlja grešku

```
class B{
      int x;
      // struktura nema default konstruktor
      B(int t){ //konstruktor sa parametrom
            x=t;
      }
}

int main(){
      B b; /* [Error] no matching function
for call to 'B::B()'*/
}
```

Konstruktor kopije

```
#include <iostream>
using namespace std;
class Tocka{
    public:
        int x,y;
        Tocka(int a, int b): x(a), y(b) {}
        Tocka(): x(0), y(0) {}
        void ispis(){cout<<"("<<x<<","<<y<<")"<<endl;}
        void postaviX(int x){this->x=x;}
        void postaviY(int y){ this->x=y;}
};
int main () {
    Tocka x(1,2),y(x);
    x.ispis();
    y.ispis();
    y.postaviX(2);
    x.ispis();
    y.ispis();
    system("pause");
    return 0;
```

Svaka klasa ima implicitni copy konstruktor koji omogućava da se napravi novi objekt kopiranjem sadržaja postojećeg objekta

Konstruktor kopije

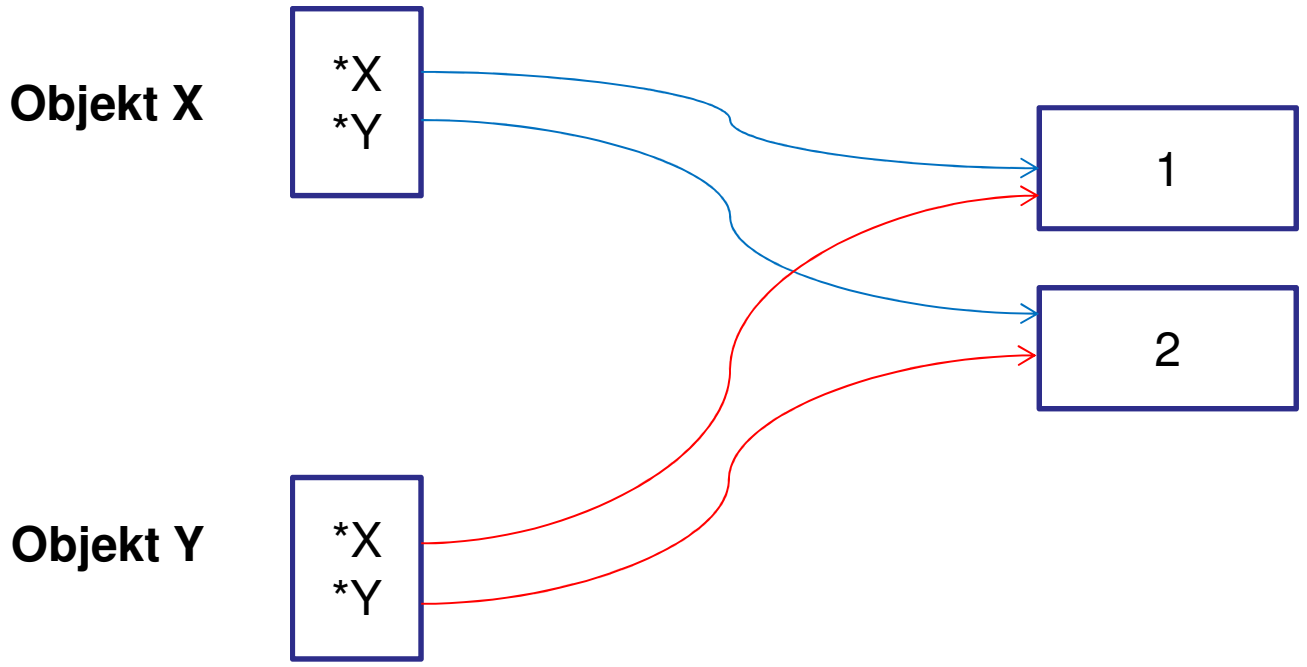
```
#include <iostream>
using namespace std;

class Tocka{
public:
        int *x;
        int *y;
        Tocka(int a=0, int b=0) {
             x=new int(a);
             y=new int(b);
        }
        ~Tocka(){
             delete x,y;
        }
        void ispis(){
             cout<<"("<<*x<<","
             <<*y<<")"<<endl;
        }
        void postaviX(int a){
             *x=a;
         }
        void postaviY(int b){
             *y=b;
         }
};
int main () {
        Tocka x(1,2),y(x);
        x.ispis();
        y.ispis();
        y.postaviX(2);
        x.ispis();
        y.ispis();
        system("pause");
        return 0;
}
```

Defaultni kopirni konstruktor ne radi kad klasa sadrži pokazivače

Konstruktor kopije

```
Tocka x(1,2),y(x); //X i Y pokazuju na iste
memorijske lokacije – nisu neovisni objekti
```



Konstruktor kopije

- Rješenje – kod klasa koje sadrže pokazivače treba kreirati dodatni copy konstruktor koji će osigurati da objekt bude ispravno kopiran (duboko kopiranje)

```
Tocka (const Tocka &T){
    x=new int(*(T.x));
    y=new int(*(T.y));
}
```

Konstruktor kopije

```
Tocka x(1,2),y(x); //ako postoji kopirni konstruktor X i Y su neovisni objekti
```