**Sveučilište u Zagrebu**
**PMF – Matematički odsjek**


# Objektno programiranje (C++)

Predavanja 06 – Kontrola kopiranja

**Vinko Petričević**

# Elementi kontrole kopiranja

- Svaki puta kada definiramo novi tip tada eksplicitno ili implicitno definiramo njegovo ponašanje prilikom kopiranja, pridruživanja i pri destrukciji. To činimo definiranjem sljedećih članova klase:
  - konstruktor kopiranjem (*copy-constructor*),
  - konstruktor premještanjem (*move-constructor*, C++11),
  - operator pridruživanja kopiranjem (*copy-assignment operator*),
  - operator pridruživanja premještanjem (*move-assignment operator*, C++11),
  - destruktor.
- Budući da objekt bilo kojeg tipa mora imati mogućnost kopiranja prevodilac će ih sintetizirati ako ih sami ne definiramo.
- Konstruktor kopiranjem ili kraće konstrutor kopije (*copy-constructor, CCtor*) je konstruktor koji kreira objekt kopiranjem objekta istog tipa. Kao argument uzima const referencu na objekt istog tipa.
- Sintetizirani CCtor će izvršiti kopiranje svih *nestatičkih* varijabli članica tako što će
  - kod ugrađenih tipova kopirati varijablu;
  - kod elemenata tipa klase pozvati njihov CCtor da obavi kopiranje (definicija je dakle rekurzivna);
  - Ako je *polje* varijabla članica onda će CCtora obaviti kopiranje polja član-po-član.

# CCtor

- Prevodilac ga koristi kada se kreira novi objekt *kopijom* istog objekta
  X a(…..) // normalni konstruktor
  X b(a) // cctor
  X c = a; // cctor
  b=a; // operator=

- Kada poziva funkciju sa parametrom koji se šalje preko vrijednosti
  f(X t);
  …
  f(a); // cctor
  // kreira se privremeni objekt kopijom varijable a, koji biva uništen nakon završetka funkcije

- Kada neka funkcija vraća neku klasu
  X g(...) {
  …
  return ret; // cctor – kreira se privremeni objekt kopijom (lokalne) varijable ret, koji biva uništen
  } // nakon što se izvrši naredba koja ga je pozvala (najvjerojatnije =)

- Pogledati primjere cctor1.cpp i cctor2.cpp

# Optimizacija prevodioca

- Što bi se dogodilo da smo pisali X b = 10?
- Prevodioc bi mogao prvo kreirati privremeni objek s parametrom 10, pa izvršiti Cctor od b s tim privremenim objektom,pa uništiti privremeni objekt
- Ili bi mogao primijeniti optimizaciju, i odmah kreirati X b(10), to se zove copy-elision
- Pogledati primjer cctor3.cpp
- Od standarda 17, ta optimizacija je obavezna

# Onemogućivanje kopiranja

- Ponekad ne želimo da se neki objekt može kopirati (konzola, datoteka, printer, …)
- Do standarda 11, to bi napravili da stavimo trivijalni cctor u privatni dio klase
- Od standarda 11, možemo pisati X (const X&)=delete
- delete se može koristiti na bilo kojoj funkciji

# Operator pridruživanja

- Operator pridruživanja mora biti funkcija članica klase. On se brine za pridruživanje objekata dane klase. Definira se tako da uzima jedan parametar koji je objekt na desnoj strani naredbe pridruživanja (desni operand); taj je parametar prirodno konstantna referenca na tip klase. Povratna vrijednost je referenca na tip klase i ona odgovara lijevom operandu — onom kojem se pridružuje.
- Ako ga ne napišemo, prevodilac ga sintetizira po pravilu kao što sintetizira i cctor
- Prevodilac će ga odbiti sintetizirati, ako klasa ima referencu, konstantni element, ili ima element kojem je obrisan ili privatan operator=

# Destruktor

- destruktor se automatski poziva
  - na završetku dosega u kojem je kreiran
  - kada pozovemo delete na objekt koji je dinamički kreiran
- destruktor bi trebao osloboditi resurse koje je varijabla zauzela (prilikom kreiranja ili za vrijeme života)
- nakon završetka izvršavanja destruktora, kompajler automatski izvrši destruktore svih nestatičkih elemenata klase

```
class C {
        D a,b
  public:
        ~C(){
              ...
        } // tu se pozovu automatski
          // pozovu destruktori od b i a
};

C *pC = new C[10];
delete[] pC; // destruktor se poziva na svaki element
```

# Upotreba

- Najčešće ne moramo ništa od ove tri funkcije mijenjati
- a ako moramo mijenjati (sami brinemo o resursima koje zauzimamo), uglavnom mijenjamo sve 3

```
class String {
        char* m_chars;
  public:
        String(const char* c = 0) {
                int len = 0;
                if (c) len = strlen(c);
                if (len) {
                        m_chars = new char[len+1];
                        strcpy(m_chars, c);
                } else m_chars=0;
        }
        const char* data() {
                return m_chars;
        }
};
```

- Kreiramo li neki String, zauzeta memorija će biti izgubljena (što možemo vidjeti pod linuxom koristeći npr. valgrind, a pod Visual Studiom <crtdgb.h>)

# Upotreba

- kompajler je automatski dodao:

```
class String {
  public:
        ~String() {}
        String(const String& s) :
              m_chars(s.m_chars) {  }
        String& operator=(const String& s) {
              m_chars = s.m_chars;
              return *this;
        }
};
```

- Primjer – string1.cpp
- međutim, klasa ne oslobodi zauzetu memoriju, pa napišimo destruktor

```
class String {
  public:
        ...
        ~String() {
              if (m_chars) delete[] m_chars;
        }
};
```

# Upotreba

- sada se memorija oslobađa dobro, ali neće dobro raditi kopiranje, ni pridruživanje
  - `String a("OP"), b(a); // b.data()[1]='5';`
  - `String c = a; c.data()[1]='6';`
  - `String d; d = a; d.data()[1]='7';`
  - `void f(String x) { d.data()[1]='8'; } f(a); // ovdje imamo 2 greške`
  - `String g() { String ret("123");return ret; } a = g();`

```
} // ovdje ćemo dobiti više grešaka
```

# Sređivanje cctora

```
class String {
...
  public:
        String(const String& s)
// : m_chars(0) // ovo kompajler stavi
  {
          int len = 0;
          if (s.m_chars)
             len = strlen(s.m_chars);
          if (len) {
            m_chars = new char[len+1];
            strcpy(m_chars, s.m_chars);
          }
          else m_chars=0;
        }
  };
```

# Sređivanje operator=

```
class String {
...
  public:
       String& operator=(const String& s) {
         if (&s == this) return *this;
         if (m_chars) delete[] m_chars;
         int len = 0;
         if (s.m_chars)
            len = strlen(s.m_chars);
         if (len) {
           m_chars = new char[len+1];
           strcpy(m_chars, s.m_chars);
         }
         else m_chars=0;
         return *this;
       }
  };
```