Sveučilište u Zagrebu
PMF – Matematički odsjek

Objektno programiranje (C++)

Predavanja 07 – 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.
- Od standarda 11, možemo eksplicitno tražiti od prevodioca da ih generira (po defaultu će biti inline osim ako je napisana van klase)

```
class Klasa {
public:
    Klasa() = default;
    Klasa(const Klasa&) = default;
    Klasa& operator=(const Klasa&);
    ~Klasa() = default;
};
Klasa& Klasa::operator=(const Klasa&) = default;
```

Zabrana kopiranja

- Prije standarda 11, ukoliko smo željeli zabraniti kopiranje, operator= i cctor bismo stavili u privatni dio klase, i eventualno i ne bismo napravili implementaciju tih funkcija (link error)
- Od standarda 11 uvedena je riječ =delete, kojom uklonimo funkciju iz klase

```
struct NoCopy {
    NoCopy() = default;
    NoCopy(const NoCopy&) = delete;
    NoCopy& operator=(const NoCopy&) = delete;
    ~NoCopy() = default;
...
};
```

- Za razliku od =default, =delete možemo koristiti i na ostalim funkcijama klase (čak i destruktor možemo obrisati, ali onda klasu nećemo moći uništiti, pa ni koristiti bez pokazivača)

Elementi kontrole kopiranja

- Kod većine klasa ne trebamo raditi ništa, sintetizirane operacije će biti dobre.
- Ukoliko klasa sadrži resurse koji ne obitavaju u klasi, moramo napraviti kopiranje i to najčešće sve 3 (5) operacija.
- Općenito prvo trebamo odrediti kako će se naša klasa ponašati
- Kao vrijednost – svaki objekt ima svoje stanje, kada ga kopiramo, kopija i original su neovisni, kada mijenjamo jednog to ne utječe na drugog (prošla predavanja).
- Kao pokazivač – kopija i original dijele podatke, pa promjene na jednom utječu na drugi – primjer brojanje referenci

swap

- Ukoliko želimo našu klasu (koja sama brine o resursima) koristiti u algoritmima koji mijenjaju redoslijed elemenata (npr. sort), bilo bi dobro napraviti funkciju swap, jer ju ti algoritmi koriste
- Po defaultu, ta funkcija bi bila
  auto temp=v1; v1=v2; v2=temp;

- Ako se vratimo na klasu string sa prethodnih predavanja, bolja implementacija bi bila da samo zamijenimo pointere:
  char* temp=v1.m_chars; v1.m_char=v2.m_chars; v2.m_chars=temp;

- Napomena: Sada bi operator= mogli napraviti i jednostavnije:

```
String& String::operator=(String s) {
    swap(*this, s);
    return *this;
}
```

Klase s varijabilnim resursima

- Nekad nam treba klasa koja pamti proizvoljan broj elemenata određene klase. Najčešće to možemo realizirati tako da koristimo vector pokazivača na elemente. Međutim ponekad nam to nije dovoljno
- Promotrimo klasu vector<string>. Kada ubacujemo novi string, ukoliko nema dosta prostora, alocira se nešto više prostora (npr. dvostruko), premjeste se svi elementi, oslobodi prethodna memorija, te se doda novi.
- Napravit ćemo jedan pojednostavljeni primjer koristeći alokatore, pomoću kojih možemo razdvojiti alokaciju od konstrukcije elemenata.

Alokatori

- Svi spremnici imaju jedan dodatni parametar predloška, tzv.alokator. Npr. std::vector je: template<class T, class Allocator = std::allocator<T> > class vector;
- On se brine o alokaciji i dealokaciji memorije (jer new odmah i inicijalizira objekt, što nam ponekad nije potrebno)
- #include <memory>
- Nekad ćemo možda i sami pisati svoje alokatore
- Alokator ima:
  - allocator<T> a // kreira alokator za tip T
  - a.allocate(n); // alocira neinicijaliziranu memoriju za n objekata tipa T; vraća pokazivač na nju
  - a.deallocate(p,n) // dealocira memoriju sa adrese p, koja sadrži prostor za n objekata tipa t. Eventualni kreirani objekti trebaju prvo biti uništeni sa destroy
  - a.construct(p, args) // na mjestu p konstrura objekt tipa T s parametrima args
  - a.destroy(p) // uništava objekt na mjestu p
- Korisne mogu biti i funkcije:
  - uninitialized_copy(b,e,b2) // kopira elemente od b do e u neinicijaliziranu memoriju b2
  - uninitialized_fill(b,e,t) // konstruira objekte u neinicijaliziranoj memoriji od b do e kopirajući t

Premještanje

- Od standarda 11, u kontrolu kopiranja dodana su dvije operacije za premještanje:
  - konstruktor premještanjem (*move-constructor*),
  - operator pridruživanja premještanjem (*move-assignment operator*),
- Kompajler ih ne sintetizira automatski.
- Oni trebaju uzeti resurse od operanda, a operand ostaviti u stanju da destruktor ne prouzroči štete
- ```
  String(String&& s) noexcept : m_chars(s.m_chars){
    //m_chars = s.m_chars;
    s.m_chars = 0;
  }
  //String(String&& s) = default;
  ```
- ```
  String& operator=(String&& s) noexcept {
    if (&s == this) return *this;
    if (m_chars) delete[] m_chars;
    m_chars = s.m_chars;
    s.m_chars = 0;
    return *this;
  }
  ```

unique_ptr<T>

- Preložak unique_ptr iz zaglavlja <memory> služi za automatsku destrukciju objekta.
- Na uništenju automatski pozove delete, ali ne radi za polja (ne pozove delete[])
- Nema operatora pridruživanja ni kopiranja, ali ima premještanje
- Ako je element klase, treba ručno napraviti kopiranje elemenata, ali ne treba destrukciju

shared_ptr<T>

- Preložak shared_ptr iz zaglavlja <memory> služi za automatsko brojenje referenci.
- Ima kopiranje
- Moguće ga je konstruirati preuzimanjem resursa od unique_ptr objekta
- Funkcija make_shared<T> kreira pametni pokazivač na zadanu vrijednost. Interno kreira objekt pomoću operatra new

- Ako ga koristimo na poljima, trebamo dodati funkciju za destrukciju

- Ako imamo cirkularnu zavisnost (A ima pokazivač na B i B na A), destrukcija se nikada neće dogoditi. U takvim slučajevima u jednoj od klasa koristimo weak_ptr<T>, koji nema vlasništvo nad objektom, ali ga može, ukoliko nije expired, dobiti pozivom funkcije lock().