POO - Constructori & Destructori

Daniel Chiș - 2020, UPB, ACS, An II, Seria AC

Subiect Email - Lab X Nume Prenume Grupă
Arhivă - Lab_X_Nume_Prenume_Grupă
Nu uitați de screenshots (-1p/ss)
Fără prezență punctajul se înjumătățește de la 9
Temă trimisă după deadline nu se punctează
Copiat - 0 puncte per temă

Constructori

Constructori - Definire

Constructorii de clase sunt funcții ce aparțin unei clase și sunt executate în momentul în care vrem să creăm un obiect al unei clase.

Un constructor va avea același nume ca și clasa din care face parte și nu returnează explicit un tip anume (nici măcar void) și poate avea oricâți parametri.

Constructorii pot fi foarte utili pentru setarea inițială a valorilor unor variabile ce aparțin clasei respective.

Există trei tipuri de constructori: default, cu parametrii și copy constructor.

Sintaxă:

```
class ClassName {

  Public: //access specifier

    …

    ClassName(parameters);  // This is the constructor

};
```

Instanțe

Un Obiect reprezintă o instanță a unei Clase. În momentul în care o clasă este definită, nu se alocă memorie, dar când se instanțiază clasa (se crează un obiect) se alocă memorie.

Sintaxă:

ClassName ObjectName;

Clasă -> Constructor -> Obiect

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

class MyClass {      // The class
  public:            // Access specifier
    MyClass() {      // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass (this will call the constructor)
  return 0;
}

```

Exemplu constructor default

Constructori cu Parametrii

Constructori cu Parametrii

Un constructor implicit (default) nu are parametrii. Dar ca și funcțiile, aceștia pot avea parametrii. Acest tip de constructori sunt utilizați în momentul în care vrem să atribuim valori unui obiect în momentul creării sale.

Sintaxă:

```
class ClassName {

public:

    Student(type parameter1, type parameter2);

};
```

```
int main(){

ClassName objectName(value1, value2);

}
```

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

class MyClass {     // The class
  public:           // Access specifier
    MyClass(string message) {     // Constructor
      cout << message;
    }
};

int main() {
  MyClass myObj("Constructor with parameter");    // Create an object of MyClass (this will
      call the constructor)
  return 0;
}
```

Exemplu 1 constructor cu parametrii

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

class Student {     // The class
public:
    string firstName;
    string lastName;
    int year;
    Student(string fn,string ln,int y) {
        firstName = fn;
        lastName = ln;
        year = y;
    }
};

int main() {
    Student student1("Andrei","Popescu",2 );
    Student student2("Maria","Moraru",2);
    cout << student1.firstName << " " << student1.lastName << " " << student1.year;
    cout << "\n";
    cout << student2.firstName << " " << student2.lastName << " " << student2.year;
    cout << "\n";
  return 0;
}
```

Exemplu 2 constructor cu parametrii

Constructori definiți în afara clasei

Ca și funcțile, contructorii pot fi definiți în afara unei clase. Mai întâi se declară constructorul în interiorul clasei, urmând ca acesta să fie definit în afara ei prin specificarea numelui clasei, urmat de operatorul :: și de numele constructorului (care e același cu cel al clasei).

Sintaxă:

```
ClassName::ConstructorName(type parameter1, type parameter2) {

....

}
```

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

class Student {     // The class
public:
    string firstName;
    string lastName;
    int year;
    Student(string fn, string ln, int y);
};

Student::Student(string fn, string ln, int y){
    firstName = fn;
    lastName = ln;
    year = y;
}

int main() {
    Student student1("Andrei","Popescu",2 );
    Student student2("Maria","Moraru",2);
    cout << student1.firstName << " " << student1.lastName << " " << student1.year;
    cout << "\n";
    cout << student2.firstName << " " << student2.lastName << " " << student2.year;
    cout << "\n";
  return 0;
}
```

Exemplu constructor definit în afara clasei

Constructori multiplii

Pentru o clasă se pot declara mai mulți constructori astfel încât să avem mai multe moduri de a instanția un obiect. Fiecare constructor poate avea parametri diferiți. În funcție de parametrii utilizați se va apela constructorul corespunzător.

```cpp
class Student {     // The class
public:
    string firstName;
    string lastName;
    int year;
    Student(string fn, string ln, int y);
    Student(string fn, int y);
    Student(int y);
    Student(){
        firstName = "Default";
        lastName = "Default";
        year = 0;
    }
};

Student::Student(string fn, string ln, int y){
    firstName = fn;
    lastName = ln;
    year = y;
}

Student::Student(string fn, int y){
    firstName = fn;
    lastName = "Dumitrescu";
    year = y;
}

Student::Student(int x){
    firstName = "Necunoscut";
    lastName = "Necunoscut";
    year = x;
}
```

```cpp
int main() {
    Student student1("Andrei","Popescu",2 );
    Student student2("Maria",2);
    Student student3(4);
    Student student4;
    cout << student1.firstName << " " << student1.lastName << " " << student1.year;
    cout << "\n";
    cout << student2.firstName << " " << student2.lastName << " " << student2.year;
    cout << "\n";
    cout << student3.firstName << " " << student3.lastName << " " << student3.year;
    cout << "\n";
    cout << student4.firstName << " " << student4.lastName << " " << student4.year;
    cout << "\n";
  return 0;
}
```

Exemplu constructuri multiplii

Copy Constructor

Copy Constructor

Copy constructor reprezintă un constructor care ia ca parametru un obiect de același tip cu clasa în care se află constructorul respectiv. Cu ajutorul acestui constructor, putem să copiem obiecte, prin copierea membru cu membru în constructor.

Dacă nu ne declarăm noi copy constructorul, compilatorul de C++ creează unul default pentru fiecare clasă. Acesta este suficient în majoritatea cazurilor, în schimb e nevoie să îl creăm noi atunci când un obiect are pointeri sau resurse alocate la runtime (fișiere, path-uri de sistem, conexiuni la internet).

Sintaxă:

```
ClassName (const ClassName &old_obj);
```

Shallow Copy vs Deep Copy

Shallow Copy Constructor - Copy Constructor Default

Deep Copy Constructor - este definit de către programator. Este folosit pentru a asigna pointerii obiectului copiat la o nouă adresă. Dacă nu am defini o nouă adresa atunci amândouă obiectele ar folosi aceeași adresă și o schimbare într-un obiect le-ar afecta pe amândouă

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

class Copy
{
private:
    char *teststring;
    int size;
public:
    Copy(const char *string = NULL); // constructor for Copy class
    ~Copy() { delete [] teststring;  }// destructor for Copy class
    Copy(const Copy&); // copy constructor for Copy class
    void print() {
        cout << teststring << endl;
    }
    void update(const char *);  // method to update the teststring
};

Copy::Copy(const char *string) //define Constructor
{
    size = strlen(string);
    teststring = new char[size+1];
    strcpy(teststring, string);
}

void Copy::update(const char *string) //define method for updating string
{
    delete [] teststring;
    size = strlen(string);
    teststring = new char[size+1];
    strcpy(teststring, string);
}
```

```cpp
Copy::Copy(const Copy& initialString) //define Copy Constructor
{
    size = initialString.size;
    teststring = new char[size+1];
    strcpy(teststring, initialString.teststring);
}

int main()
{
    Copy str1("Initial");
    Copy str2 = str1;

    str1.print();
    str2.print();

    str2.update("Updated");

    str1.print();
    str2.print();
    return 0;
}
```

Exemplu deep copy constructor

Destructori

Destructori

Destructorul este o funcție specială în cadrul unei clase care este executată în momentul în care un obiect al clasei respective nu mai are nici un scop. Acesta distruge/șterge obiectul.

Destructorul are același nume ca și clasa, având în față operatorul ~. Nu poate returna nici o valoare și nici nu poate primi parametrii. Destructorii sunt utilizați pentru a elibera resursele utilizate (memoria). Apelarea se realizează astfel: NumeObiect.~Destructor();

Destructorul este apelat automat în momentul în care se termină o funcție sau programul.

Fiecare clasă are un singur destructor.

Destructorii sunt creați automat dacă nu îi creăm noi, iar aceștia merg bine atâta timp cât nu avem memorie alocată dinamic. Dacă o clasă conține un pointer către o zonă de memorie atunci trebuie să folosim un destructor pentru a elibera memoria înainte ca instanța clasei să fie distrusă. Dacă nu se realizează această operație pot apărea pierderi de memorie (memory leak).

```
#include <iostream>
#include <string>
using namespace std;
class Message {
   public:
    string message;
    void setLength( double len );
    double getLength( void );
    Message(string message);   // This is the constructor declaration
    ~Message();  // This is the destructor: declaration
};

// Member functions definitions including constructor
Message::Message(string message) {
    cout << "Object is being created" << endl;
    cout << message.length() <<endl;
}
Message::~Message(void) {
    cout << "Object is being deleted" << endl;
}

// Main function for the program
int main() {
   Message message("Automatica");

   // set line length

   return 0;
}
```

Exemplu destructori

Diferență față de Java...

Având în vedere că în Java nu există pointeri, tocmai pentru a ușura munca programatorilor, nu există nici destructori.

În Java există o componentă numită **Garbage Collector** care șterge toate obiectele care nu mai sunt folosite, în mod automat.

Nice to read: https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Summary

De Reținut

Constructorul - metodă utilizată pentru instanțierea unui obiect, apelat în momentul creării acestuia

Reguli constructori - numele este același ca și cel al clasei și nu are return type

Tipuri de constructori - default, cu parametrii și copy

Copy constructor - default (shallow) și creat de programator(deep), mereu există doar unul într-o clasă și e folosit pentru a crea obiecte noi din unul existent

Destructor - metodă utilizată pentru a distruge un obiect, folosește același nume ca cel al clasei + ~

Exerciții

Exerciții

1. Să se implementeze o clasă cu constructor default care să afișeze data curentă (ziua și ora) în momentul instanțierii. *Tip: check chrono library.* **3p**
2. Să se implementeze o clasă *Car*. Această clasă trebuie să aibă constructori multiplii (minim 4). Constructorii trebuie să fie definiți în afara clasei. Clasa *Car* trebuie să aibă minim 4 atribute și un constructor să fie de tipul default. Creați un obiect pentru fiecare constructor. **4p**
3. Să se implementeze o clasă *Sibling*. Atributele din interiorul clasei vor fi cele de nume, prenume și vârstă, toate alocate prin intermediul pointerilor. Pentru această clasă trebuie să aveți și un copy constructor. Creați un obiect *sibling1*. Creați un obiect *sibling2* cu ajutorul copy constructorului și faceți update la prenume și vârstă. Afișați atributele fiecărui obiect. **2p**

*toate exercițiile trebuie să aibă și destructorii definiți