物件導向程式設計

Constructor & Destructor

Joseph Chuang-Chieh Lin
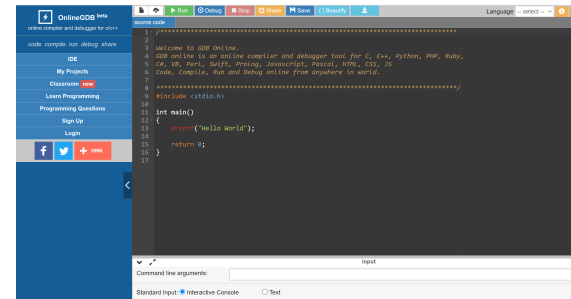Dept. CSIE, Tamkang University

Platform

- Dev-C++

  Click here to download.
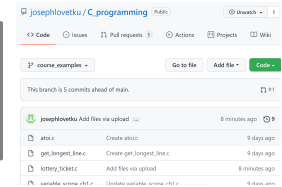
  Note: Please use this version otherwise you can't compile your programs/projects in Win10.

- OnlineGDB (https://www.onlinegdb.com/)

  My GitHub page:
  click the link here to visit.

- Other resources:
  - MIT OpenCourseWare - Introduction to C++ [link].
  - Learning C++ Programming [Programiz].
  - GeeksforGeeks [link]

Constructors

- Each class defines how objects of its type can be initialized.

- Classes control object initialization by defining one or more special member functions known as **constructors**.

- The job of a constructor is to **initialize the data members of a class object**.

- A constructor is run whenever an object of a class type is created.

- It's very useful for setting initial values for certain member variables.

Constructors

- The compiler generates a default constructor, called **synthesized default constructor**, automatically only if a class declares no constructors.

- **Note:** for some classes, the synthesized default constructor does the wrong thing.

Constructors

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

class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line(); // the constructor
   private:
      double length;
};
```

```cpp
Line::Line(void) {
    cout << "Object is being created\n";
}
void Line::setLength(double len) {
    length = len;
}
double Line::getLength(void) {
    return length;
}
```

```cpp
int main() {
   Line line;

   line.setLength(6.0); // set line length
   cout << "Length of line : " << line.getLength() <<endl;

   return 0;
}
```

```
Object is being created
Length of line : 6
```

5

Constructors

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

class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line() = default;
      // synthesized default constructor
   private:
      double length;
};
```

```cpp
void Line::setLength(double len) {
    length = len;
}
double Line::getLength(void) {
    return length;
}
```

```cpp
int main() {
   Line line;

   line.setLength(6.0); // set line length
   cout << "Length of line : " << line.getLength() <<endl;

   return 0;
}
```

```
Length of line : 6
```

6

Parameterized Constructors

Refer to: https://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm

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

class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line(double len);
      // constructor with parameters
   private:
      double length;
};
```

```cpp
Line::Line(double len) {
    cout << "Object is being created, "
         << "length = " << len << endl;
    length = len;
}
void Line::setLength(double len) {
    length = len;
}
double Line::getLength(void) {
    return length;
}
```

```cpp
int main() {
   Line line(10.0);

   cout << "Length of line : " << line.getLength() <<endl;
   line.setLength(6.0);
   cout << "Length of line : " << line.getLength() <<endl;
   return 0;
}
```

```
Object is being created, length = 10
Length of line : 10
Length of line : 6
```

7

Constructor Initializer List

Refer to: https://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm

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

class Line {
   public:
      void setLength( double len );
      double getLength(void);
      Line() = default;
      Line(double len): length(len) {};
   private:
      double length;
};
```

```cpp
void Line::setLength(double len) {
    length = len;
}
double Line::getLength(void) {
    return length;
}
```

```
Length of line1: 4.68426e-310
Length of line2: 10
Length of line1: 6
```

```cpp
int main() {
   Line line1, line2(10.0);
   cout << "Length of line1: " << line1.getLength() <<endl;
   cout << "Length of line2: " << line2.getLength() <<endl;
   line1.setLength(6.0);
   cout << "Length of line1: " << line1.getLength() <<endl;
   return 0;
}
```

Destructors

- **Destructors** do whatever work is needed to **free** the resources used by an object and **destroy** the **nonstatic data members** of the object.

- The destructor is a member function with the name of the class prefixed by a tilde **(~)**.

- It has no return value and takes no parameters.

  - Cannot be overloaded.

  - There is always only one destructor for a given class.

```
class Foo {
public:
    ~Foo(); // destructor
// ...
};
```

Destructors

- A destructor also has a function body and a destruction part.

- In a destructor:
    - The function body is executed first, and then the members are destroyed.
    - Members are destroyed in reverse order from the order in which they were initialized.

- The function body of a destructor does whatever operations the class designer wishes to have executed subsequent to the last use of an object.
    - Typically, the destructor **frees resources** an object allocated during its lifetime.

- The destruction part is implicit.
    - What happens when a member is destroyed depends on the type of the member.
    - Members of class type are destroyed by running the member's own destructor.

- The built-in types do not have destructors, so nothing is done to destroy members of built-in type.

Destructor Examples

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

class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line();    // constructor
      ~Line();   // destructor

   private:
      double length;
};
```

```cpp
Line::Line(void) {
    cout << "Object is being created"
         << endl;
}
Line::~Line(void) {
    cout << "Object is being deleted"
         << endl;
}
void Line::setLength(double len) {
    length = len;
}
double Line::getLength(void) {
    return length;
}
```

```cpp
int main() {
   Line line;
   line.setLength(6.0);
   cout << "Length of line : "
        << line.getLength() << endl;
   return 0;
}
```

```
Object is being created
Length of line : 6
Object is being deleted
```

11

Destructor Examples

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

class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line();   // constructor
      ~Line() { cout << "An object with length " << length << " is out!" << endl;} ;

   private:
      double length;
};
```

```cpp
void Line::setLength(double len) {
    length = len;
}
double Line::getLength(void) {
    return length;
}
```

```cpp
int main() {
   Line line1, line2(10.0);
   cout << "Length of line1: " << line1.getLength() << endl;
   cout << "Length of line2: " << line2.getLength() << endl;
   line1.setLength(6.0);
   cout << "Length of line1: " << line1.getLength() <<endl;
   return 0;
}
```

```
Length of line2: 10
Length of line1: 6
An object with length 10 is out!
An object with length 6 is out!
```

12

Class Exercise (2%)

- Add constructor(s) and a destructor to the following class

```cpp
class rectangle {
public:
    typedef int unit;
    void area();
    void set(unit wd, unit ht);
private:
    unit width;
    unit height;
};
```

```cpp
void rectangle::set(unit wd, unit ht)
{
    width = wd;
    height = ht;
}
```

```cpp
void rectangle::area()
{
    cout << "The area: " << width * height << endl;
}
```

```cpp
int main() // DO NOT modify main()
{
    rectangle obj, obj2(2,5); //creating object of rectangle class
    rectangle::unit x, y;
    cin >> x;
    cin >> y;
    obj.set(x, y);
    obj.area();
    obj2.area();
    return 0;
}
```

13

Copy Constructor

- Reference:
  - https://courses.cs.washington.edu/courses/cse333/12su/lectures/lec11.pdf
    - Page 8~

- An Example of "Person":
  - https://onlinegdb.com/8EeWdA3zv

Another Example
-from cplusplus.com

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

class Example {
    string* ptr;
  public:
    // constructors:
    Example(): ptr(new string) {}
    Example (const string& str): ptr(new string(str)) {}
    // destructor:
    // since we dynamically allocate a string
    ~Example () {delete ptr;}
    // access content:
    const string& content() const {return *ptr;}
};

int main () {
  Example foo;
  Example bar ("TKU CSIE IS THE BEST!");
  cout << "bar's content: " << bar.content() << '\n';
  return 0;
}
```

wan

More on the Copy Constructor

```
MyClass::MyClass (const MyClass&);
```

If a class has no custom copy nor move constructors (or assignments) defined, an *implicit copy constructor* is provided.

This copy constructor simply performs a copy of its own members.
For example,

```
class MyClass {
public:
    int a, b;
    string c;
};
```

An implicit copy constructor is automatically defined and is equivalent to

```
MyClass::MyClass(const MyClass& x):
a(x.a), b(x.b), c(x.c)  {}
```

When is the copy constructor called?

```
MyClass foo;
MyClass bar (foo);     // object initialization: copy constructor called
MyClass baz = foo;     // object initialization: copy constructor called
foo = bar;             // object already initialized: copy assignment called
```

```
MyClass& operator= (const MyClass& x) {
    delete ptr;
    ptr = new string (x.content());
    return *this;
}
```

Operator overloaded
(We will discuss about it in the future.

Another Example (Destructor + Copy Constructor)
-from cplusplus.com

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

class Example {
    string* ptr;
  public:
    // constructors:
    Example(): ptr(new string) {}
    Example (const string& str): ptr(new string(str)) {}
    Example (const Example& x): ptr(new string(x.content())) {}
    // destructor:
    ~Example () {delete ptr;}
    // access content:
    const string& content() const {return *ptr;}
};

void main () {
  Example foo("TKU CSIE IS THE BEST!");
  Example bar = foo;

  cout << "bar's content: " << bar.content() << '\n';
}
```

*Move Constructor

```
MyClass (MyClass&&);                // move-constructor
MyClass& operator= (MyClass&&);   // move-assignment
```

- Similar to copying, moving also uses the value of an object to set the value to another object.
- But, unlike copying, the content is actually transferred from one object (the source) to the other (the destination):
  - **The source loses that content**, which is taken over by the destination.
  - This moving only happens when the source of the value is an ***unnamed*** object.

```
MyClass fn();            // function returning a MyClass object
MyClass foo;             // default constructor
MyClass bar = foo;       // copy constructor
MyClass baz = fn();      // move constructor
foo = bar;               // copy assignment
baz = MyClass();         // move assignment
```

Exercise 04 (3%)

- Matrix data member + constructors + destructor

```cpp
class Matrix_Widget {
private:
    int mat[3][3];
public:
    Matrix_Widget() = default;
    Matrix_Widget(int v);
    // initialize mat to be v at each position
    void modify(int i, int j, int value);
    void showMatrix();
};
```

```cpp
int main()
{
    Matrix_Widget obj, obj2(1);
    obj.showMatrix();
    obj2.showMatrix();
    obj.modify(2, 1, 7);
    Matrix_Widget obj3(obj);
    obj3.showMatrix();
    return 0;
}
```

Exercise 04 (3%)

- Matrix data member + constructors + destructor

- Requirement:

  - **Add a parameterized constructor and a copy constructor.**

  - **Implement the member function:**
    `Matrix_Widget::modify(int i, int j, int v);`

    - The value at `(i,j)` of the matrix `mat` is changed to `v`.

  - Implement the function **`Matrix_Widget::showMatrix()`** and the **destructor** such that the program with the following main() function results in the sample input & output.

Exercise 04 (3%)

- Matrix data member + constructors + destructor

Sample output

```
0 0 0
0 0 0
0 0 0

1 1 1
1 1 1
1 1 1

0 0 0
0 0 0
0 7 0

obj deleted
obj deleted
obj deleted
```