Constructors and Destructors

Chapter 4

Objectives

- **Explain the use of constructors in C++ and describe their benefits.**
- **Use constructors in your programs to initialize member data and perform other initializations.**
- **Describe the use of multiple constructors in a class, including the default constructor.**
- **Describe destructors and use them in your programs.**
- **Explain how there can be “hidden” constructors in a C++ program.**
- **Simplify a class by using default arguments in a constructor.**
- **Gain experience through code walk-throughs and lab exercises.**

  - The example programs are in the **chapter directory.**
  - Labs located in Labs/Lab4

The Problem of Initialization

- **A classical problem in computer programming is ensuring that variables are properly initialized.**
  - Requires a mechanism that ensures that initialization code is automatically called when the object is created.

Constructors and Initialization

- **C++ provides a mechanism, known as a *constructor*, to facilitate initialization of objects.**
  - A constructor is invoked (instantiated) automatically whenever an object is created.
  - It is the programmer's responsibility to provide the proper constructor, which involves a prototype in the class definition and code in the class implementation.
  - If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).

Constructor != function

- A constructor is like a special member function having the same name as the class.
- A constructor is *not* a function, as it does not have a return type.
- A constructor can have arguments and can be overloaded.

```
class IntStack{
public:
     IntStack();          // constructor prototype
     ...
};
IntStack::IntStack()  {  // constructor implementation
    ...
}
```

Object Creation and Destruction

- **The constructor is invoked when an object is created, e.g. by a specific declaration:**
  - IntStack stack; // stack is a new IntStack instance
- **Constructors are also invoked in several less obvious ways such as:**
  - When the compiler creates a temporary object (e.g. in call-by-value function invocation),
  - When an object is created dynamically by **new**.
  - More on this later…
- **An object may be destroyed implicitly:**
  - When it goes “out of scope” (e.g. a local variable in a function).
  - When a compiler temporary variable is destroyed.
- **An object may be destroyed explicitly:**
  - An object created dynamically by **new** will be destroyed by **delete** (to be discussed later in chapter on memory management).

Destructors

- **Whenever an object is destroyed, a destructor is called, if one is defined for the class.**
- **A destructor is like a member function, whose name is the class name preceded by a tilde (~).**
- **A destructor takes no arguments, and so there can only be one destructor**

```
class String {
public:
  ~String();  // destructor prototype...
};
String::~String(){
 //  destructor implementation
}
```

Multiple Constructors

- **A class can have several constructors, each having a different parameter list signature, just as ordinary functions can be overloaded in C++.**
- **The constructor taking no arguments is called the *default constructor*.**

```
const int STRINGSIZE = 80;
class String {
public:
    String();  // default constructor
    String(const char *str);
    ~String();
private:
    char m_str[STRINGSIZE];
};
```

Demo

- **Open the folder String.**
- **Create a project containing the files Strn.cpp, Strn.h, and the test program DemoStrn.cpp.**
- **Build and run the program.**
- **Running the code shows four more invocations of destructors than there are of constructors! Why?**

Hidden Constructors

- **There is a “hidden” constructor at work, called the copy constructor, which we will study in a later chapter:**
  - The **PrintStrings()** function passes its parameters by value, which causes a copy of the parameters to be made
  - When a compiler temporary variable is destroyed, the destructor is invoked just as it is for objects explicitly created by the programmer’s code

Using Default Arguments

- **A constructor can be simplified by using a default argument in the second constructor.**

```
const int STRINGSIZE = 80;
class String
{
public:
    String(const char *str = “”);
    ~String();
    void SetString(const char *str);
    const char *GetString();
private:
    char m_str[STRINGSIZE];
};
```

Summary

- **Constructors are used to initialize member data and perform other initializations.**
- **Constructors are automatically called when an object is created.**
- **The default constructor has no arguments, and through overloading a class can have other constructors.**
- **The destructor of a class is called when an object is destroyed.**
- **There can be “hidden” constructors in a C++ program, invoked, for example, when an object is copied in a call-by-value function invocation.**
- **A class can be simplified by using default arguments in a constructor.**