Unit 6 Initialization and clean up

Section 1 Guaranteed initialization with the constructor

* Define the term constructor.

If a class has a constructor, the compiler automatically calls that constructor at the point an object is created, before client programmers can get their hands on the object. The constructor call isn’t even an option for the client programmer; it is performed by the compiler at the point the object is defined.

Here’s a simple class with a constructor:

class X {

int i;

public:

X(); // Constructor

} ;

Constructors and destructors return nothing, and you don’t have an option.

Section 2 Guaranteed Cleanup with the Destructor

* Define the term destructor.

If you just forget about it, your object never achieves closure upon its exit from this world. In C++, cleanup is as important as initialization and is therefore guaranteed with the destructor.

he syntax for the destructor is similar to that for the constructor: the class name is used for the name of the function. However, the destructor is distinguished from the constructor by a leading tilde (~). In addition, the destructor never has any arguments because destruction never needs any options. Here’s the declaration for a destructor

See example Constructor in folder

Section 3 Elimination of the definition Block

* Define the term definition block.

In general, C++ will not allow you to create an object before you have the initialization information for the constructor.

* List at least two reasons to dispense with the definition block.

By reducing the duration of the variable’s availability within the scope, you are reducing the chance it will be misused in some other part of the scope.

In addition, readability is improved because the reader doesn’t have to jump back and forth to the beginning of the scope to know the type of a variable.

* Describe the lifetime of loop variables.

The variables **i** and **j** are defined directly inside the **for** expression (which you cannot do in C). They are then available for use in the **for** loop. It’s a very convenient syntax because the context removes all question about the purpose of **i** and **j**, so you don’t need to use such ungainly names as **i\_loop\_counter** for clarity.

Lifetime is until which point the variable exists until it is out of scope or the variable whiting the brace braket