Chapter 2. Variables and Basic Types

***2.4. const Qualifier***

const Objects Are Local to a File By Default. Nonconst variables are extern by default. To make a const variable accessible to other files we must explicitly specify that it is extern.

***2.5. References***

A const reference can be initialized to an object of a different type or to an rvalue, such as a literal constant:

int i = 42;

// legal for const references only

const int &r = 42;

const int &r2 = r + i;

The same initializations are not legal for nonconst references. Rather, they result in compile-time errors. The reason is subtle and warrants an explanation.

This behavior is easiest to understand when we look at what happens when we bind a reference to an object of a different type. If we write

double dval = 3.14;

const int &ri = dval;

the compiler transforms this code into something like this:

int temp = dval; // create temporary int from the double

const int &ri = temp; // bind ri to that temporary

If ri were not const, then we could assign a new value to ri. Doing so would not change dval but would instead change temp. To the programmer expecting that assignments to ri would change dval, it would appear that the change did not work. Allowing only const references to be bound to values requiring temporaries avoids the problem entirely because a const reference is read-only.

My Codes:

int main(int argc, char\* argv[])

{

int aa = 10;

const int &ref\_aa = aa;

double dd = 89.983;

const int &ref\_dd = dd;

int dd\_aa = dd;

cout << ref\_aa << endl;

cout << ref\_dd << endl;

cout << dd\_aa << endl;

cout << endl;

cout << "&aa = " << &aa << endl;

cout << "&ref\_aa = " << &ref\_aa << endl;

cout << "&dd = " << &dd << endl;

cout << "&ref\_dd = " << &ref\_dd << endl;

system("PAUSE");

return 0;

}

Output:

10

89

89

&aa = 0018FC24

&ref\_aa = 0018FC24

&dd = 0018FC08

&ref\_dd = 0018FBF0

***2.8. Class Types***

C++ supports a second keyword, [struct](mk:@MSITStore:D:\\Zhipeng\\books\\c++primer.chm::/0201721481/ch02lev1sec11.html" \l "gloss02_53), that can be used to define class types. The struct keyword is inherited from C.

If we define a class using the class keyword, then any members defined before the first access label are implicitly private; ifwe usethe struct keyword, then those members are public. Whether we define a class using the class keyword or the struct keyword affects only the default initial access level.

***2.9. Writing Our Own Header Files***

When designing a header it is essential to remember the difference between definitions, which may only occur once, and declarations, which may occur multiple times ([Section 2.3.5](mk:@MSITStore:D:\Zhipeng\books\c++primer.chm::/0201721481/ch02lev1sec3.html#ch02lev2sec14), p. [52](mk:@MSITStore:D:\Zhipeng\books\c++primer.chm::/0201721481/ch02lev1sec3.html#ch02lev2sec14)). The following statements are definitions and therefore should not appear in a header:

extern int ival = 10; // initializer, so it's a definition

double fica\_rate; // no extern, so it's a definition

Although ival is declared extern, it has an initializer, which means this statement is a definition. Similarly, the declaration of fica\_rate, although it does not have an initializer, is a definition because the extern keyword is absent. Including either of these definitions in two or more files of the same program will result in a linker error complaining about multiple definitions

There are three exceptions to the rule that headers should not contain definitions: classes, const objects whose value is known at compile time, and inline functions are all defined in headers.

**A Brief Introduction to the Preprocessor**

A #include directive takes a single argument: the name of a header. The pre-processor replaces each #include by the contents of the specified header. Our own headers are stored in files. System headers may be stored in a compiler-specific format that is more efficient.

Avoiding Multiple Inclusions

#ifndef SALESITEM\_H

#define SALESITEM\_H

// Definition of Sales\_itemclass and related functions goes here

#endif