C++ Primer Notes

mingzailao

2016-9-11

Contents

1 Begin

1.1 Writing a Simple C++ Program

1.1.1 Example 1.1

```
#include<iostream>
using std::cout;
using std::endl;
int main()
{
    cout<<"Hello world"<<endl;
    return 0;
}
```

1.1.2 Compiling and Executing Our Program

1. Comliling

   ```
   #!/bin/bash
   cd Code
   g++ hello.cpp -o Hello
   ```

   In the next, I just use Automake to compile.

2. Executing ./Hello

1.1.3 Exercise

1. Exercise 1.2: Change the program to return -1. A return value of -1 is often treated as an indicator that the program failed. Recompile and rerun your program to see how your system treats a failure indicator from main.

   (a) Answer

```
#include<iostream>

int main()
{
    return -1;
}
```

1.2 A First Look at Input/Output

1.2.1 Standard Input and Output Objects

The library defines four IO objects:

1. istream:cin
2. ostream:cout
3. cerr
4. clog

1.2.2 A Program That Uses the IO Library

```
#include<iostream>
int main()
{
    std::cout<<"Enter two numbers : "<<std::endl;
    int v1=0,v2=0;
    std::cin>>v1>>v2;
    std::cout<<"The sum of "<<v1<<"and "<<v2
        <<" is "<<v1+v2<<std::endl;
    return 0;
}
```

1.2.3 Writing to a Stream

```
std::cout<<"Enter two number"<<std::endl;
```

The << operator takes two operands: The left-hand operand must be an ostream object(std::cout); the right-hand operand is a value to print. The operator writes the given value on the given ostream

1. Notes for std::endl Writing endl has the effect of ending the current line and flushing the buffer.

1.2.4 Using Names from the Standard Library

The prefix std:: indicates that the names cout and endl are defined inside the namespace named std.

1.2.5 Reading from a Stream

```
std::cin >> v1 >> v2;
```

The input operator (the >> operator) behaves analogously to the output operator. It takes an istream as its left-hand operand and an object as its right-hand operand. It reads data from the given istream and stores what was read in the given object. The input operator returns its left-hand operand as its result.

1. Notes All the operators just like a function which can return its left-hand operand or its right-hand operand.

1.2.6 Completing the Program

```
std::cout << "The sum of " << v1 << " and " << v2
<< " is " << v1 + v2 << std::endl;
```

It prints each of its operands on the standard output. What is interesting in this example is that the operands are not all the same kinds of values. Some operands are string literals, such as "The sum of ". Others are int values, such as v1, v2, and the result of evaluating the arithmetic expression v1 + v2. The library defines versions of the input and output operators that handle operands of each of these differing types.

1.2.7 EXercise

1. Exercise 1.3: Write a program to print Hello, World on the standard output.

   (a) Answer

   ```
   #include<iostream>
   int main()
   {
       std::cout<<"Hello world"<<std::endl;
       return 0;
   }
   ```

2. Exercise 1.4: Our program used the addition operator, +, to add two numbers. Write a program that uses the multiplication operator, *, to print the product instead.

   (a) Answer

   ```
   #include<iostream>
   int main()
   {
       std::cout<<"Enter two numbers : "<<std::endl;
       int v1=0,v2=0;
       std::cin>>v1>>v2;
       std::cout<<"The product of "<<v1<<"and "<<v2
           <<" is "<<v1*v2<<std::endl;
       return 0;
   }
   ```

3. Exercise 1.5: We wrote the output in one large statement. Rewrite the program to use a separate statement to print each operand.

   (a) Answer

   ```
   #include<iostream>
   int main()
   {
       std::cout<<"Enter two numbers : "<<std::endl;
       int v1=0,v2=0;
       std::cin>>v1>>v2;
       std::cout<<"The sum of "<<std::endl;
       std::cout<<v1<<std::endl;
   ```

```
        std::cout<<"and "<<std::endl;
        std::cout<<v2<<std::endl;
        std::cout<<"is "<<std::endl;
        std::cout<<v1+v2<<std::endl;
        return 0;
    }
```

1.3 A Word about Comments

1.3.1 Kinds of Comments in C++

1. //
2. * *

1.4 Flow of Control

1.4.1 The while Statement

1. Example

```
#include <iostream>
int main()
{
    int sum=0,val =1;
    while(val<=10){
        sum+=val;
        ++val;
    }
    std::cout<<"Sum of 1 to 10 is : "<<sum<<std::endl;
    return 0;
}
```

2. Notes

`val+=1`

val+=1 ⇔ val=val+1 ~~val ⇔ val=val+1 and the value of ++val is val(after) val~~ ⇔ val=val+1 and the value of val++ is val(before)

3. Exercise

(a) Exercise 1.9: Write a program that uses a while to sum the numbers from 50 to 100.

i. Answer

```
int sum(int a,int b)
{
    int result=0;
    for(int i=a;i<=b;i++){
        result+=i;
    }
    return result;
}
#include <iostream>
#include "ex_1_9.hpp"
int main()
{
    std::cout<<"Sum of 50 to 100 is : "<<sum(50,100)<<std::endl;
    return 0;
}
```

(b) Exercise 1.10: In addition to the ++ operator that adds 1 to its operand, there is a decrement operator (–) that subtracts 1. Use the decrement operator to write a while that prints the numbers from ten down to zero.

i. Answer

```
#include<iostream>
int main()
{
    for(int i=10;i>=0;i--){
        std::cout<<i<<" ";
        std::cout<<std::endl;
    }
    return 0;
}
```

(c) Exercise 1.11: Write a program that prompts the user for two integers. Print each number in the range specified by those two integers.

i. Answer

```
#include<iostream>
#include "ex_1_9.hpp"
```

```
int main()
{
    int a,b;
    std::cin>>a>>b;
    std::cout<<sum(a,b)<<std::endl;
    return 0;
}
```

1.4.2 The for Statement

1. Example

```
#include<iostream>
int main()
{
    int sum=0;
    for(int val=1;val<=10;++val){
        sum+=val;
    }
    std::cout<<"Sum of 1 to 10 is :"<<sum<<std::endl;
    return 0;
}
```

1.4.3 Reading an Unknown Number of Inputs

1. Example

```
#include<iostream>
int main()
{
    int sum = 0, value = 0;
    while (std::cin >> value)
        sum += value;
    std::cout << "Sum is: " << sum << std::endl;
    return 0;
}
```

2. Notes When you do not flushing the buffer, the std::cin will get your inputs. (in MacOS, ctrl+d)

3. Exercise

(a) Exercise 1.16: Write your own version of a program that prints the sum of a set of integers read from cin.

i. Answer

```
#include<iostream>
int main()
{
    int result=0,item;
    while(std::cin>>item){
        result+=item;
        std::cout<<result<<std::endl;
    }
    return 0;
}
```

1.4.4 The if Statement

1. Example

```
#include<iostream>
int main()
{
    int currVal=0,val=0;
    if(std::cin>>currVal){
        int cnt=1;
        while(std::cin>>val){
            if(val==currVal)
                ++cnt;
            else{
                std::cout<<currVal<<" occurs "
                         <<cnt<<" times "<<std::endl;
                currVal=val;
                cnt=1;
            }
        }
        std::cout<<currVal<<" occurs "
                 <<cnt<<" times"<<std::endl;
    }
    return 0;
}
```

1.5 Introducing Classes

1.5.1 The *Sales_item* Class

```
Sales_item item
```

1. Notes item is an object of Class *Sales_item*, it own the items and the function of *Sales_item*
2. Interface
   - Call a function named isbn to fetch the item isbn from a *Sales_item* object.
   - Use the input (») and output («) operators to read and write objects of type *Sales_item*.
   - Use the assignment operator (=) to assign one *Sales_item* object to another.
   - Use the addition operator (+) to add two *Sales_item* objects. The two objects must refer to the same ISBN. The result is a new $\text{Sales}_{\text{item}}$ object whose ISBN is that of its operands and whose number sold and revenue are the sum of the corresponding values in its operands.
   - Use the compound assignment operator (+=) to add one *Sales_item* object into another.
3. Exercise
   (a) Exercise 1.20: `http://www.informit.com/title/032174113` contains a copy of *Sales_item*.h in the Chapter 1 code directory. Copy that file to your working directory. Use it to write a program that reads a set of book sales transactions, writing each transaction to the standard output.
      i. Answer

```
#include <iostream>
#include "Sales_item.h"
using std::cin;
using std::cout;
using std::endl;

int main()
{
```

```
    for (Sales_item item; cin >> item; cout << item << endl);
    return 0;
}
```

(b) Exercise 1.21: Write a program that reads two *Sales_item* objects that have the same ISBN and produces their sum.

i. Answer

```
#include <iostream>
#include "Sales_item.h"

using std::cin;
using std::cout;
using std::endl;
using std::cerr;

int main()
{
    Sales_item item1, item2;
    cin >> item1 >> item2;
    if (item1.isbn() == item2.isbn())
        cout << item1 + item2 << endl;
    else
        cerr << "Different ISBN." << endl;
}
```

(c) Exercise 1.22: Write a program that reads several transactions for the same ISBN. Write the sum of all the transactions that were read.

i. Answer

```
#include <iostream>
#include "Sales_item.h"

int main()
{
    Sales_item total;
    if (std::cin >> total)
    {
        Sales_item trans;
        while (std::cin >> trans)
        {
            if (total.isbn() == trans.isbn())
```

```
                    total += trans;
                else
                {
                    std::cout << total << std::endl;
                    total = trans;
                }
            }
            std::cout << total << std::endl;
        }
        else
        {
            std::cerr << "No data?!" << std::endl;
            return -1;
        }

        return 0;
    }
```

1.5.2 A First Look at Member Functions

1. Example

```
#include<iostream>
#include "Sales_item.h"

int main()
{
    Sales_item item1,item2;
    std::cin>>item1>>item2;
    if(item1.isbn()==item2.isbn()){
        std::cout<<item1+item2<<std::endl;
        return 0;
    }
    else{
        std::cerr<<"Data must refer to the same ISBN"
                 <<std::endl;
        return -1;
    }
}
```

2. Notes

```
item1.isbn() == item2.isbn()
```

uses the dot operator (the . operator) to say that we want the isbn member of the object named item1.

3. Exercise

   (a) Exercise 1.23: Write a program that reads several transactions and counts how many transactions occur for each ISBN.

      i. Answer

```
#include <iostream>
#include "Sales_item.h"

int main()
{
    Sales_item currItem, valItem;
    if (std::cin >> currItem)
    {
        int cnt = 1;
        while (std::cin >> valItem)
        {
            if (valItem.isbn() == currItem.isbn())
            {
                ++cnt;
            }
            else
            {
                std::cout << currItem
                          << " occurs "
                          << cnt
                          << " times "
                          << std::endl;
                currItem = valItem;
                cnt = 1;
            }
        }
        std::cout << currItem
                  << " occurs "
                  << cnt
                  << " times "
```

```
                                << std::endl;
                }
                return 0;
            }
```

1.6 The Bookstore Program

```
#include<iostream>
#include "Sales_item.h"
int main()
{
    Sales_item total;
    if(std::cin>>total){
        Sales_item trans;
        while(std::cin>>trans){
            if(total.isbn()==trans.isbn())
                total+=trans;
            else{
                std::cout<<total<<std::endl;
                total=trans;
            }
        }
        std::cout<<total<<std::endl;
    }
    else{
        std::cerr<<"No data?!"<<std::endl;
    }
    return 0;
}
```

2 Variables and Basic Types

2.1 Primitive Built-in Types

C++ defines a set of primitive types that include the arithmetic types and a special type named void.

2.1.1 Arithmetic Types

The arithmetic types are divided into two categories

- integral types (which include character and boolean types)
- floating-point types.

Type	Meaning	Minimum Size
bool	boolean	NA
char	charactor	8 bits
wchar_t	wide charactor	16 bits
*char*16*_t*	Unicode charactor	16 bits
*char*32*_t*	Unicode charactor	32 bits
short	short integer	16 bits
int	integer	16 bits
long	long integer	32 bits
long long	long integer	64 bits
float	single-precision floating-point	6 significant digits
double	double-precision floating-point	10 significant digits
long double	extended-precision floating-point	10 significant digits

1. Notes(Signed and Unsigned Types) Except for bool and the extended character types, the integral types may be signed or unsigned. A signed type represents negative or positive numbers (including zero); an unsigned type represents only values greater than or equal to zero.

2. Exercise

   (a) Exercise 2.1: What are the differences between int, long, long long, and short? Between an unsigned and a signed type? Between a float and a double?

      i. Answer

         - Use int for integer arithmetic. short is usually too small and, in practice, long often has the same size as int. If your data values are larger than the minimum guaranteed size of an int, then use long long. (In a word: short $<$ int $<$ long $<$ long long)
         - Use an unsigned type when you know that the values cannot be negative. (In a word: no negative, unsigned.)
         - Use double for floating-point computations; float usually does not have enough precision, and the cost of double-precision calculations versus single-precision is negligible. In fact, on some machines, double-precision operations

are faster than single. The precision offered by long double usually is unnecessary and often entails considerable run-time cost. (In a word: float $<$ double $<$ long double)

(b) Exercise 2.2: To calculate a mortgage payment, what types would you use for the rate, principal, and payment? Explain why you selected each type.

i. Answer use double, or also float.
The rate most like that: 4.50 % per year. The principal most like that: 854.36 The payment most like that: 1, 142.36

2.1.2 Literals

1. Notes(Character and Character String Literals)

```
'a'     // character literal
"Hello world!"    //string literal
```

2.2 Variables

2.2.1 Variable Definitions

```
int sum=0,value,
    units_sold=0;
Sales_item item;
std::string book("0-201-78345-X");
```

1. Initializers An object that is initialized gets the specified value at the moment it is created. The values used to initialize a variable can be arbitrarily complicated expressions.

2. List Initialization

```
int units_sold = 0;
int units_sold = {0};
int units_sold{0};
int units_sold(0);
```

3. Default Initialization

(a) Notes For class,Each class controls how we initialize objects of that class type. Key: Construct Function

2.2.2 Variable Declarations and Definitions

A declaration makes a name known to the program. A file that wants to use a name defined elsewhere includes a declaration for that name. A definition creates the associated entity.

To obtain a declaration that is not also a definition, we add the extern keyword and may not provide an explicit initializer:

```
extern int i; // declares but does not define i
int j; // declares and defines j
```

in a function, if you want to initialize a variable marked with extren, you will get an error.

2.2.3 Identifiers

Identifiers in C++ can be composed of letters, digits, and the underscore character. The language imposes no limit on name length. Identifiers must begin with either a letter or an underscore. Identifiers are case-sensitive; upper- and lowercase letters are distinct:

```
// defines four different int variables
int somename, someName, SomeName, SOMENAME;
```

2.3 Compound Types

A compound type is a type that is defined in terms of another type. C++ has several compound types, two of whichreferences and pointerswell cover in this chapter.

2.3.1 References

A reference defines an alternative name for an object.

A reference type refers to another type.

```
int ival = 1024;
int &refVal = ival; // refVal refers to (is another name for) ival
int &refVal2; // error: a reference must be initialized
```

A reference is not an object

2.3.2 Pointers

A Pointers is an object(unlike reference),so Pointers can be assigned and copied;a single pointer can point to several different objects over its lifetime.

A pointer need not be initialized at the time it is defined.

Like other built-in types, pointers defined at block scope have undefined value if they are not initialized.

```
int *ip1,*ip2;\\ ip1 and ip2 are both pointers to int
double dp,*dp2; \\ dp2 is a pointer to double and dp is an object of double
```

1. Taking the Address of an Object A pointer holds the address of another object. We get the address of an object by usin the address-of operator (the & operator):

   ```
   int ival=42;
   int *p=&ival; // p is an pointer object to ival(int)
   ```

2. Using a Pointer to Access an Object When a pointer points to an object, we can use the dereference operator (the * operator) to access that object:

   ```
   int ival=42;
   int *p=&ival; // p is a pointer object to ival(int object)
   cout<<*p // use * get the object to which p points
   ```

3. Assignment and Pointers Both pointers and references give indirect access to other objects. However, there are important differences in how they do so. The most important is that a reference is not an object. Once we have defined a reference, there is no way to make that reference refer to a different object. When we use a reference, we always get the object to which the reference was initially bound. There is no such identity between a pointer and the address that it holds. As with any other (nonreference) variable, when we assign to a pointer, we give the pointer itself a new value. Assignment makes the pointer point to a different object:

   The important thing to keep in mind is that assignment changes its left-hand operand.

   ```
   pi=&ival// value in pi is changed; pi now points to ival
   *pi=0 // value in ival is changed; pi is unchanged
   ```

4. void* Pointers The type void* is a special pointer type that can hold the address of any object. Like any other pointer, a void* pointer holds an address, but the type of the object at that address is unknown:

```
double obj = 3.14, *pd = &obj;
// ok: void* can hold the address value of any data pointer type
void *pv = &obj; // obj can be an object of any type
pv = pd; // pv can hold a pointer to any type
```

2.3.3 Understanding Compound Type Declarations

It is a common misconception to think that the type modifier (* or &) applies to all the variables defined in a single statement. Part of the problem arises because we can put whitespace between the type modifier and the name being declared:

```
int* p//legal but might be misleading
int* p1, p2; // p1 is a pointer to int; p2 is an int
int *p1, *p2; // both p1 and p2 are pointers to int
```

so we should avoid int* but use int *