C and C + + Language Application Similarity Analysis

Zhou Jianru

Department of Information Engineering, Sichuan Information Technology College

GuangYuan, 628040

Keywords: C + + language; Pointer; Reference; String

Abstract. Although C language and C + + are two different programming languages, they have some similarities in grammar and subtle differences in practical application. This article summarizes practical experience. Combined with practical application, it analyzes tiny differences of related technical key points in C and C + + in detail in Boolean data type, string handling, pointers and parameter passing referred in programming, illustrates the reasons of differences, and puts forward a proposal for the application.

C is a process-oriented language, while C + + is an object-oriented language. There are a lot of common characteristics between the two: a variety of data types, operators, flexible operation, etc. The two languages have similarities in basic grammar and subtle differences in application. With application examples, C and C + + logical value calculation, string handling, pointers and references are analyzed in detail.

The essence analysis of C + + Boolean type

C language has no Boolean type, while C + + language has Boolean type. Although there is no Boolean type in C language, there are logic operations and relationship operations, and also the concepts of logic and logical false [1]. 1 is presented as logic true and 0 means logic false. Although there is Boolean type in C + + language, it is not real Boolean logic values true and false, but with very similar way in C language: the value of 1 and 0 for logical true and false. Because logical value can be expressed as 1 and 0, so there is no essential difference between logic and relationship of C and C + + with flexible application. The following are detailed analysis of procedures.

Sample program 1

```
#include <iostream>
using namespace std;
int main()
{
      Boolean x=-2,y=0;
      int a=-1,b=0,c=1;
cout<<(8>=3)+2-(2&&-1)<<endl;  ①
cout<<(2*x)<<endl;  ②
cout<<(x>y)<<endl;  ③
      cout<<(a<b<c)<<endl;    ④
       cout<<(a&&b||c)<<endl;  ⑤
      return 0;
}
```

Due to the value of Boolean type variable in c + + language is 1 (true) or 0 (false), so Boolean variables can not only make logic and relationship calculation, but can perform arithmetic operations. The statement in program 1 (1), for example, is logic value involved arithmetic. Relationship calculation 8 > = 3 result is logical value 1, and logic operation results of 2 && - 1 is 1, so the entire expression result for 1+2-1 is integer 2. In addition, give Boolean variable a nonzero integer (non-zero is true logic), but when Boolean variable is involved in operation, its value is not the assigned integer, but 1. Because it means logic "true", so it is taken as 1 for operation. Take program 1 statements (2) and (3) for example. Despite the variable assignment is x - 2, 2×x is 2×1 which equals to 2, not 2×-2 which equals to -4. Relational expression x > y is equivalent to 1 > 0, so the relationship is established, and the result is "true" value of 1, not 0.

The relationship and logic operation between C and C + + language are basically different from Java language. Statements (4) and (5), for example, the result of the expression a<b<c，a&&b||c using c + + algorithm is 0, but in Java language, it is a wrong expression. The Boolean type values in Java language are true and false, but not an integer value 1 and 0. Because of different data types, they can't have numerical computation. Java language expression a < b < c has to be written as a<b&&b<c to be right. Expression a&&b||c completely does not comply with grammar rules, because a, b, c are integer variables, while operand participated in Java logic operations must be logic value of true or false. The analysis of application 1 can help to understand C and C + + language differences and flexibilities in the related calculation of logical value.

C string and C + + string class differences in mixed application

C + + language integrates string (C string) function in C language, and also expands string class string, so, when programming with C + + language, C string library functions can be used, and string class member function can be used. When form parameter of self-defined function is string class object, and the actual parameter is C character array, both types are different. We need to convert C character array into C + + string class string object. How should they be handled? Please see the following sample program analysis.

Sample program 2

```
# include < cstrings > / / C string library functions
# include < string > / / c + + string class function
# include < iostream >
Using namespace STD.
Int main ()
{
Char ps1 [10] = "China";
Char ps2 [10] = "sichuan";
Char ps [20] = " 0";
String s1 = string (ps1); (1) / / in C character array structure string object s1
String s2 = string (ps2);
S1 + = s2; (2) / / c + + string concatenation
Strcpy (ps, s2. C_str ()); (3) / / C string is copied to ps array
Strcat (ps, ps1); (4) / / C string concatenation
Return 0;
}
```

The standard c + + string class constructor function is string (const char * s) which can use C language string or character array to build C + + string class object (i.e., C + + string class string), for example, the statement (1) in program 2 is this use. The string class object in c + + language can use the overloaded operator "+" for the connection of the strings, such as the statement (2) in program 2. Connect string s2 at the end of string s1, and do not consider whether the strings s1 can accommodate all characters or not, because string class object is equivalent to dynamic array, and its memory bytes will increase with string length automatically. C language string concatenation, however, cannot use "+" operator connection, and can only use strcat () function, such as statement (4), premise condition is the enough length of array ps for all the characters of string ps1.

C_str () function prototype is const char * c_str (); It is a member function of string class. Return a null terminated string and return the first address of the current string, but it can't directly assign to the address to the character type pointer. We can copy the current string to an appropriate character array length, such as statement (3). S2 is string class object, s2. C_str () is to convert C + + string objects s2 to C string constants, and copy the string constants by C language copy function to the appropriate length string array ps.

When programming C + + language, C string, library functions and string class and member function can be used mixed, but both have substantial distinctions. C string is common data type, either a character array (char STR [10]), either a pointer to string (char *str). Once memory allocated, its length cannot automatically change. Pay attention to the length of the string in connection string and copy operation. C + + string class string is class, which has rich member functions, and its use format is object name, function name (); When mixed operate C string and string objects, it is best to make types into a consistency as far as possible to avoid direct object calculation of C string and string class.

Different applications of Pointers and references

By return statements, function can only return one value, but in actual application, what should we do to return two or more values? Whether it can be done through function arguments? Yes, but when function call occurs, arguments and formal parameters are of value transfer, which is unable to realize return value, because arguments and formal parameters belong to local variables of different functions. The system assigns different memory spaces for them and there is no connection between them, so, after function call parameter, they cannot return the results to the arguments. Reference type variable in c + + language can realize function return value, and reference type variables are mainly used for function parameters or function return value [2].

Pointer as function parameters passes pointers variables address, which copies the value of argument pointer (variables address) to parameter pointer. At this point, argument and function parameter point to the same location, but they are two different pointer variables. If changes parameter pointer pointing, it does not affect the argument pointer's pointing.

When a reference type variable is taken as function parameter to pass, the argument passes itself to the function parameter, rather than a copy of the argument, namely the argument and the function parameter are the same address unit, therefore, modifies the function parameter and also changes the argument, so the way of parameter passing saves the space and time. The process of parameter passing of pointer variables and reference variables is shown in the following procedure.

```
void fun(int *r)
{
   int x=100;
   r=&x;
   cout<<r<<endl;
}
int main(    )
{
   int a=0,*p=&a;
   fun(p);
   if(p==&a)
   cout<<"p point to variable a "<<p<<endl;
   return 0;
}
```

```
void test(int *&r)
{
   int x=100;
   r=&x;
   cout<<r<<endl;
}
int main( )
{
   int a=0,*p=&a;
   test(p);
   if(p!=&a)
      cout<<" p does not point to a variable a
"<<p<<endl;
   return 0;
}
```

Reference

[1] Zhou Jianru. C language logic relationship and logical operation analysis [J]. Journal of Electronic Test, 2012 (22) : 38-39.

[2] Zhou Zhide, Hou Zhengchang. C + + program design [M]. Beijing: Electronic Industry Press, 2005.

[3] Eckel, Liu Zongtian. C + + programming ideas [M]. Beijing: Mechanical Industry Publishing House, 2012.

Author introduction

Zhou Jianru (1980 -), male, Han ethnic group, Shaanxi Chenggu, master of engineering, Lecturer in Sichuan Information Technology College, research direction: software engineering