

C++ Object Oriented Programming

VLSI design is getting more and more important these days. The sizes of the circuits and devices are getting larger and larger. In the old days people relied on using CAD tools to do the layout of the whole circuits. Later on people started to use tools like Verilog or VHDL to implement the algorithms in a design in a more natural way, and then use synthesis tools and place and route tools to do the layout of the chips.

The trend of designing chips are getting higher and higher level. People now think that using Verilog or VHDL is still not very efficient. They want to directly use the implementation in C or C++, and transform it into Verilog code using some kind of CAD tools. Programs and libraries like Ocapl started to come out, which provide a way for people to convert a certain C++ implementation of a chip to VHDL or Verilog. Therefore it is also very important for hardware designers to know about C and C++.

1.0 Overview

Most of you should know about C already, and this document mostly describes a key features in C++ programming, object oriented programming (OOP). A very brief introduction to C++ syntax, which is different from C, is described first. Then we will go in more depth about how to use objects in our programs. We will talk about why we want to use C++ rather than C, the syntax on how to describe classes and objects in C++, and also we will talk about how to declare and use an object in C++.

2.0 Basic Syntax

In C++, if you want to output a variable `temp` to the screen, you do:
`cout << "value of temp is " << temp << "\n";`

In C++, if you want to read a value into variable `temp`, you do:
`cin >> temp;`

In C++, in order to use `cin` and `cout`, you have to include the library `<iostream.h>`. The method for implementing file I/O is also different from C, and you can look into the reference for the correct syntax and what header file to include. For some functions like `rand()` you can still use them by including the header file `<stdlib.h>`.

Other than these, C++ is almost the same as C. So if you already know about C, learning about C++ shouldn't be an extremely challenging task. Making function calls, assignment, return format, loops, arrays, are all the same as C.

All C++ files used in this class have `.cxx` as extension. For example you have a C++ file called `temp.cxx` you can use the following command to compile it under UNIX (both SUN or HP):

```
g++ temp.cxx
```

And you can execute the program by:

```
./a.out
```

3.0 Why C++ Not C

The main reason why we want to use rather than C++ is that we want to take advantage of the ability to describe an object in C++. In C you can write the following:

```
struct BankAccount {  
    int accountno;  
    int amount;  
    int interestrate;  
};
```

And you can start declaring variables as type `BankAccount`. However, you cannot include any functions in this type. For example if a variable `account` of this type is declared to describe a bank account for a user, and the user should be able to deposit and show account info. What can you do? All you can do is to write a function to access `account.amount` or `account.accountno`. You cannot have a function that you can call and directly ask for the amount you want to deposit and the info you want to show.

This leads to the issue of privacy. For example, if `interestrate` is something you don't want your user to know how it is generated. There is no way in C using `struct` that you can do that. User can always access the variable `interestrate`.

And these are the problems that C++ aims at. It allows you to actually describe an object in real life. For example you want to have an object called `BankAccount`, and it will contain variables `accountno`, `amount`, `interestrate`. You want to have the functions for user to open up an account, make some deposit, and calculate the interest. You also want a secret function that will let help you to calculate the interest rate, where user shouldn't know. Last of all, you only want the user to access those variables through the designed functions, and not just by referring them directly.

You can do ALL these in C++. You can design it in such a way that there is an interface for each object with the outside world, and the user can't access those data that you want to be private.

4.0 Object Oriented Programming

By using classes in C++, you are able to describe an object as thorough as you want, and thus able to implement what we see in real life. For example in real life when you go out and see a car, you consider it as an object. You know you can accelerate through putting more gas. You won't care about how it accelerates. For example when you open a bank account, you consider that as another object. You know you get an account number, and you can do deposit, ask for your account info. You cannot go and directly change the amount, nor can you change your account number, without going through the teller.

Basic Class Syntax

For example you want to declare a class called *BankAccount*.

```
class BankAccount {  
    public:  
        // you can put all the functions that you can allow the user to  
        // access. This can be called as the interface to the outside world.  
    private:  
        // this is where you put all your "secret" components. The data or  
        // variables that you don't want people to have access to. You can  
        // store those functions that you don't want people to know here.  
};
```

4.1 Constructor

Inside public:

```
BankAccount(int dollars, int accountno);  
BankAccount(int accountno);
```

The constructor function (after class definition):

```
BankAccount::BankAccount(int dollars, int accountno);  
BankAccount::BankAccount(int accountno);
```

A constructor is a member function that is automatically called when an object of that class is declared. Inside a constructor function you will do all the initialization you want on the data members. For example you can assign the account number, put in the amount of money, etc. Notice that you are allowed to have more than one constructor. This is called overloading of functions. Therefore if you have more than one constructor in your class you can have more than one method to declare an object of that kind.

4.2 Data

Having data fields in class is just the same as that in struct in C. They have exactly the same syntax. For example if you have the variables *accountno*, *amount*, and *interestrate*, where all of them you don't want user to have access to, you will put these statements under private:

```
int accountno;  
int amount;  
int interestrate;
```

4.3 Member Functions

As mentioned before, you can have functions inside class that can provide an interface with the outside world. These functions can be either under public or private, and their syntax are the same. For example if you have an function to display info of an account called *display*, and a user should be able to access that function, the following line would be added under the public area:

```
void display (accountno);
```

And use the following syntax to implement the function:

```
void BankAccount::Display(int accountno)  
{  
.....;  
.....;  
}
```

Just like constructor, you can also overload any member functions.

4.4 Overloading Operators

Another fancy thing about C++ is that you can overload operators. Imagine you are writing a C program, and you have two string, *c1* and *c2*, and you want to compare if they are the same. Can you do the following?

```
if (c1 == c2)
```

Good luck. You can't because the operator "==" is not defined for string. For the same reason if you have two accounts *a1* and *a2* and you want to compared them, you CANNOT just plainly use the following statement:

```
if (a1 == a2)
```

However you can do that if you overload the operator “==” so that it can do whatever you tell it to do. First you will put this statement under public:

```
friend int operator ==(BankAccount& a1, BankAccount& a2);
```

Then you will implement the overloading function by adding this function after the class:

```
int operator ==(BankAccount& a1, BankAccount& a2)  
{  
.....;  
.....;  
if (.....) return 1;  
else return 0;  
}
```

This way you can start using the operator “==” in your program.

4.5 Class access

After you have finished implementing your object, you can start using it. For example you want to declare an object of type BankAccount called account, you can do the following:

```
BankAccount account(12345);
```

And you can access the functions / operators by:

```
account.display(12345);  
if (account1 == account2) cout << "you are rich";
```

5.0 Conclusion

Now with all these information you should be able to write a C++ program using classes. All the syntax in writing classes are provided, and these are just for referenced. If you want to learn more about them, you have to go to some other references.

6.0 References

- Walter J. Savitch, “Problem Solving With C++ : The Object of Programming”, 1998.
Harvey M. Deitel, Paul J. Deitel, “C++ How to Program (How to Program Series)”, 1997.