

Introduction to a C++ low level object model

 codeproject.com/Articles/468882/Introduction-to-a-Cplusplus-low-level-object-model



Sergey Vystoropskiy

Rate
this:



4.80 (22 votes)



7 Oct 2012 CPOL

C++ low level design

Introduction

In this article I am going to describe how does a C++ compiler interprets an objects. Also I am going to describe some interesting cases of language behaviour. For some people this article may look like trivial. But I do believe that it might be usefull for a big number of C++ developers.

What is struct ?

First of all let's think how does compiler looks at struct. Suppose we have a struct:

Hide Copy Code

```

struct SomeStruct
{
    int field1;
    char field2
    double field3;
    bool field4;
};

```

And suppose we have a function

Hide Copy Code

```

void SomeFunction()
{
    SomeStruct someStructVariable;
    // usage of someStructVariable
    ...
}

```

How do you think what will happen if we will replace the someStructVariable into a for variables like below ?

Hide Copy Code

```

void SomeFunction()
{
    int field1;
    char field2
    double field3;
    bool field4;
    // usage of 4 variables
    ...
}

```

Correct answer is: nothing ! The machine code generated for this function stays the same. So for compiler struct is just a couple variables placed one by one. So the pointer to the struct is actually a pointer to the first member of struct. So the code like this:

Hide Copy Code

```
someStructPtr->field1 = 123;
```

will be interpreted like:

Hide Copy Code

```
*(int*)((void*)someStructPtr) = 123;
```

and a code:

Hide Copy Code

```
someStructPtr->field3 = 0.123;
```

will be

Hide Copy Code

```
*((double*)((char*)someStructPtr + sizeof(int) + sizeof(char))) = 0.123;
```

and so on ...

It's true until struct is just a simple set of fields without virtual functions. Such kind of structs are called POD structs.

Structs's with member functions.

Suppose we had add a member function to our struct definition:

Hide Copy Code

```
struct SomeStruct
{
    int field1;
    char field2;
    double field3;
    bool field4;
    void SetAllFields(int f1, char f2, double f3, bool f4)
    {
        field1 = f1;
        field2 = f2;
        field3 = f3;
        field4 = f4;
    }
};
```

How do you think what is the number of arguments of SetAllFields function ? The correct answer is five. Because for all member functions we have a hidden argument "this". This hidden parameter allows compiler to have only one copy of member function code and use it for all class instances. So for compiler our member function will look like:

Hide Copy Code

```
void SetAllFields(SomeStruct* this, int f1, char f2, double f3, bool f4)
{
    this->field1 = f1;
    this->field2 = f2;
    this->field3 = f3;
    this->field4 = f4;
}
```

So calling of member function like this:

Hide Copy Code

```
someStructInstance.SetAllFields(123, 'a', 0.123, true);
```

will be transformed in:

Hide Copy Code

```
SetAllFields(&someStructInstance, 123, 'a', 0.123, true);
```

By the way according to Strastrup the only difference between class and struct is that all members of struct are public by default, and class members are private by default. So we can say that all I have mentioned is true for a classes.

Looking ahead we can take a look on a standart interview question:

Suppose we have a class SomeClass1 and class a SomeClass2 derived from SomeClass1. Both of them have a function PrintClassName wich prints a class name.

Hide Copy Code

```

...
class SomeClass1
{
public:
    void Print()
    {
        std::cout << "SomeClass1\n";
    }
};

class SomeClass2 : public SomeClass1
{
public:
    void Print()
    {
        std::cout << "SomeClass2\n";
    }
};
...

```

It's obvious that the following code:

Hide Copy Code

```

...
SomeClass1* ptr = new SomeClass2();
ptr->Print();
...

```

will print "SomeClass1" and now we can explain why. Call of the print function will be interpreted like the code below.

Hide Copy Code

```

...
SomeClass* ptr = new SomeClass2();
Print(ptr);
...

```

So the compiler used the best matched function, as it does not actually knows what kind of object "ptr" points on.

Inheritance realization.

We have already used inheritance in this article, but we did not explain the mechanisms that are covered behind it.

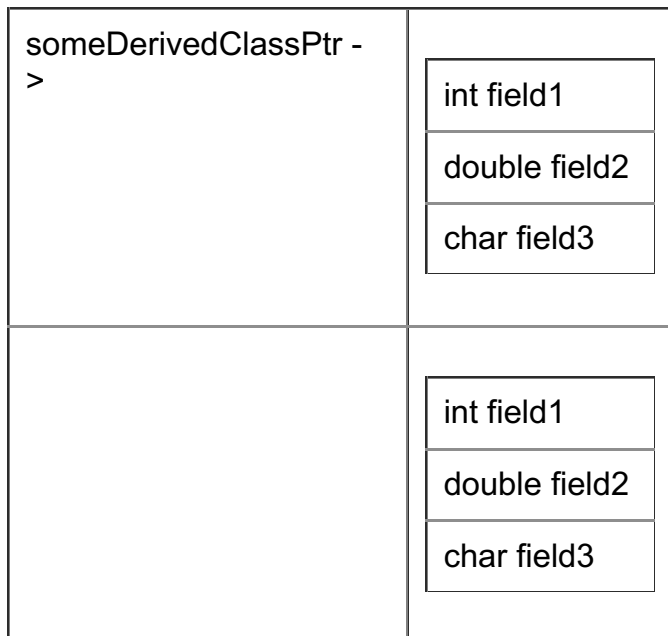
Suppose we have a 2 classes: class SomeClass and class SomeDerivedClass.

Hide Copy Code

```
...
class SomeClass
{
public:
    int field1;
    double field2;
    char field3;
};

class SomeDerivedClass : public SomeClass
{
public:
    int field1;
    double field2;
    char field3;
};
...
```

As we all know the instance of `SomeDerivedClass` will contain a fields both from base class and from derived class. So The most obvious way to place all fields in memory is just place base class members, then place a derrved class members. That's actually the way how it's implemented in C++ compiler.



In case when we have a deeper inheritance nothing changes. We have base class at first, then derived from base than the derived from derived from base and so on...

Multiple inheritance does not make a lot of changes in object representation. In case of inheritance from 2 classes, the fields of the first ancestor will be placed first, after that there will be a fields of the second ancestor and then the fields of derived class.

Diamond inheritance problem.

Suppose we have some base class "BaseClass", and two classes are derived from it: "SomeClass1" and "SomeClass2". Then we had created one more class "SomeFinalClass" which is inherited from "SomeClass1" and "SomeClass2".

Hide Copy Code

```
...
class BaseClass
{
...
};
class SomeClass1 : public BaseClass
{
...
};
class SomeClass2 : public BaseClass
{
...
};
class SomeFinalClass : public SomeClass1, public SomeClass2
{
...
};
...
```

The struct of that class in memory will be the following:

BaseClass members
SomeClass1 members
BaseClass members
SomeClass2 members
SomeFinalClass members

As usual we will need only one copy of base class members, so we need to avoid the second copy, because it's additional memory usage and calling BaseClass constructor twice.

To avoid that we can use a virtual inheritance. In this case we will have only one copy of base class members and only one base class constructor call.

Virtual functions realization.

Let's come back to the last example from the "Structs's with member functions." chapter, but with a little modification. We will put virtual keyword before our functions definition.

Hide Copy Code

```
...
class SomeClass1
{
public:
    virtual void Print()
    {
        std::cout << "SomeClass1\n";
    }
};
class SomeClass2 : public SomeClass1
{
public:
    virtual void Print()
    {
        std::cout << "SomeClass2\n";
    }
};
...
```

The output in a following code will differ from the previous one.

Hide Copy Code

```
...
SomeClass1* ptr = new SomeClass2();
ptr->Print();
...
```

It will be "SomeClass2". So it's obvious that mechanism of calling virtual functions should be different.

All classes with virtual functions has a special hidden member which is a pointer to a virtual functions table, So calls of virtual functions are calls of function by pointer with some offset.

So lets take a look on the following code.

Hide Copy Code

```
...
SomeClass1* ptr = new SomeClass1();
ptr->Print();
...
```


In machine codes we will have something like this:

Hide Copy Code

```
...  
//class initialization  
...  
ptr->vftbl[0](ptr);  
...
```

You probably interesting why do we create an instance of SomeClass1 instead of SomeClass2 ? This is because the thing getting interesting when we are trying to call a virtual function from a derived class. In a case of inheritance a new virtual table is created on a base of an old one. All overloaded functions from the base class are overloaded with the according functions of an ancestor. Virtual functions which was not present in a parent class are append to a virtual table in order they were declared.

Now lets take a look on a class initialization. As we know for derived classes construction starts from calling base class constructor(s) and then the derived class constructor. Base class constructor sets the vftbl pointer to a virtual table of base class. Then in an inherited class constructor vftbl pointer is reseted to a derived class table. So that's why when we called a virtual function like in an example at the begining of this paragraph we get an output "SomeClass2". vftbl pointer was pointing to another virtual table where were another function by the offset 0.

I should mention that virtual functions mechanizm is not working in a constructor and destructor. As for the constructor it's because vftbl pointer is resetted in the end of construction. It's made like that because while constructor is now finished we should think that the fields of a class are not initialized. As an example the following code will print "SomeClass1"

Hide Copy Code

```

class SomeClass1
{
public:
    virtual void Print()
    {
        std::cout << "SomeClass1\n";
    }
};
class SomeClass2 : public SomeClass1
{
public:
    SomeClass2()
    {
        Print();
    }
    virtual void Print()
    {
        std::cout << "SomeClass2\n";
    }
};
...
SomeClass1* ptr = new SomeClass2();
...

```

As for a destructor it also resets a vftbl but it does at the start, so the base class function will be called.

NOTE: it is not necessary to put virtual for all function definitions. If you will put virtual once all the function's with the same signature and from the same inheritance tree will be virtual.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

Share

About the Author

Sergey Vystoropskiy

Software Developer (Senior) Oracle

United States 



No Biography provided





Comments and Discussions

You must **Sign In** to use this message board.


Spacing Layout Per page




 My vote of 5  gndnet 7-Nov-12 8:30




 My vote of 5  serhiy.semenyuk 18-Oct-12 23:42




 this dos not happen on MSVC  jmanuelexe 9-Oct-12 10:06




 Re: this dos not happen on MSVC   Sergey Vystoropskiy 28-Oct-12 11:48




 [My vote of 1] My vote - bad   SergV-SNV 7-Oct-12 21:20

 Re: [My vote of 1] My vote - bad   Sergey Vystoropskiy 7-Oct-12 21:42

 Re: [My vote of 1] My vote - bad   SergV-SNV 7-Oct-12 22:29

 Re: [My vote of 1] My vote - bad   Sergey Vystoropskiy 7-Oct-12 22:43

 Re: [My vote of 1] My vote - bad   SergV-SNV 7-Oct-12 22:55

 Re: [My vote of 1] My vote - bad   Espen Harlinn 8-Oct-12 0:31

 Re: [My vote of 1] My vote - bad   Sergey Vystoropskiy 8-Oct-12 0:50



Re: [My vote of 1] My vote - bad



SergV-SNV

8-Oct-12 1:32

Last Visit: 1-Nov-20 22:55
18:36

Last Update: 2-Nov-20

Refresh 1



General



News



Suggestion



Question



Bug



Answer



Joke



Praise



Rant



Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads,
Ctrl+Shift+Left/Right to switch pages.

[Go to top](#)