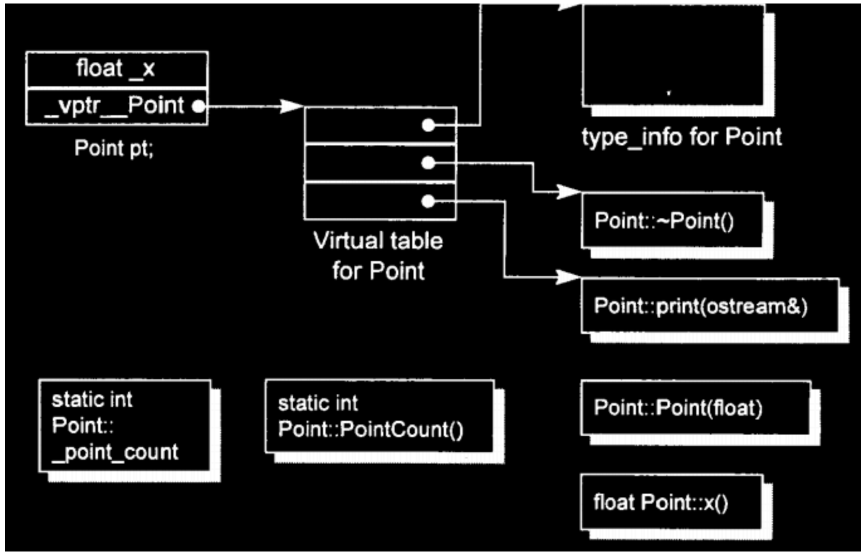


Coroutine in C Language
Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities
Regex C++
Using std::map Wisely With Modern C++

Email Print WhatsApp LinkedIn Reddit Twitter Facebook Messenger

Reading Time: 6 minutes
In this article, we will see the memory layout of different C++ Object. And how different storage & access specifiers affect this memory footprint. I am not going to discuss compiler augmented code, name mangling & working of any C++ mechanism related to memory as it is compiler & architecture-dependent. To keep this further simple, I have considered the standard stack growth direction i.e. downwards. And with the same, data members are represented inverted in memory(thanks to Thomas Vermeilh for pointing that out), that too I have ignored for simplicity. Although if you want to play around with the same, you can refer to this link.

So, in nutshell, it's just all about ***"How different objects are laid out in memory?"***

Contents [hide]

SCROLL

```
class X {
    int     x;
    float   xx;

public:
    X() {}
    ~X() {}

    void printInt() {}
    void printFloat() {}
};
```

- Memory layout:

```
        |                        |
        |------------------------| <------ X class object memory layout
        |        int X::x        |
        |------------------------|  stack segment
        |       float X::xx      |       |
        |------------------------|       |
        |                        |      \|/
        |                        |
        |                        |
------|------------------------|----------------
        |         X::X()         |
        |------------------------|       |
        |        X::~X()         |       |
        |------------------------|      \|/
        |      X::printInt()     |  text segment
        |------------------------|
        |     X::printFloat()    |
        |------------------------|
        |                        |
```

- In conclusion of the above example, only data members got the space in the stack. That too as the same order of their declarations. Because this is guaranteed by most of the compilers, apparently.
- In addition, all other methods, constructor, destructor & compiler augmented code goes into the text segment. These methods are then called & passed `this` pointer implicitly of calling object in its 1st argument which we have already discussed in Inside The C++ Object Model.

Coroutine in C Language

Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities

Regex C++

Using std::map Wisely With Modern C++

SCROLL

```cpp
public:
    X() {}
    virtual ~X() {}

    virtual void printAll() {}
    void printInt() {}
    void printFloat() {}
    static void printCount() {}
};
```

- Memory layout:

```
      |                        |
      |------------------------| <------ X class object memory layout
      |        int X::x        |
stack |------------------------|
  |   |       float X::xx      |
  |   |------------------------|      |-------|--------------------------|
  |   |        X::_vptr        |------|       |        type_info X       |
 \|/  |------------------------|              |--------------------------|
      |           o            |              |    address of X::~X()    |
      |           o            |              |--------------------------|
      |           o            |              | address of X::printAll() |
      |                        |              |--------------------------|
      |                        |
------|------------------------|------------
      |  static int X::count   |      /|\
      |------------------------|       |
      |           o            |  data segment
      |           o            |       |
      |                        |      \|/
------|------------------------|------------
      |        X::X()          |
      |------------------------|       |
      |        X::~X()         |       |
      |------------------------|       |
      |      X::printAll()     |      \|/
      |------------------------|  text segment
      |      X::printInt()     |
      |------------------------|
      |     X::printFloat()    |
```

- Coroutine in C Language
- Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities
- Regex C++
- Using std::map Wisely With Modern C++

SCROLL

- And static data member got the space into the data segment of memory. Which access with scope resolution operator(i.e. `::`). But after compilation, there is nothing like scope & namespace. Because, its just name mangling performed by the compiler, everything referred by its absolute or relative address. You can read more about name mangling in C++ here.
- Moreover, static methods go into the text segment and called with the scope resolution operator. Except for `this` pointer which is not passed in its argument.
- For virtual keyword, the compiler automatically inserts pointer(`_vptr`) to a virtual table into the object memory representation. So it transforms direct function calling in an indirect call(i.e. dynamic dispatch which I have discussed in How Does Virtual Function Works Internally?). Usually, virtual table created statically per class in the data segment, but it also depends on compiler implementation.
- In a virtual table, 1st entry points to a `type_info` object which contains information related to current class & DAG(Directed Acyclic Graph) of other base classes if it is derived from them.
- I have not mentioned the data type of `_vptr` because the standard does not mention(even I don't know that).

Layout of C++ Object With Inheritance

```
class X {
    int     x;
    string str;

public:
    X() {}
    virtual ~X() {}

    virtual void printAll() {}
};

class Y : public X {
    int     y;

public:
    Y() {}
    ~Y() {}

    void printAll() {}
};
```

Coroutine in C Language

Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities

Regex C++

Using std::map Wisely With Modern C++

SCROLL

```
   |    |        char* string::str         |
  \|/   |----------------------------------|      |-------|---------------------------|
        |          X::_vptr                |------|       |        type_info Y        |
        |----------------------------------|              |---------------------------|
        |         int Y::y                 |              |    address of Y::~Y()     |
        |----------------------------------|              |---------------------------|
        |              o                   |              | address of Y::printAll()  |
        |              o                   |              |---------------------------|
        |              o                   |
  ------|----------------------------------|--------
        |          X::X()                  |
        |----------------------------------|        |
        |          X::~X()                 |        |
        |----------------------------------|        |
        |        X::printAll()             |       \|/
        |----------------------------------|  text segment
        |          Y::Y()                  |
        |----------------------------------|
        |          Y::~Y()                 |
        |----------------------------------|
        |        Y::printAll()             |
        |----------------------------------|
        |     string::string()             |
        |----------------------------------|
        |     string::~string()            |
        |----------------------------------|
        |     string::length()             |
        |----------------------------------|
        |              o                   |
        |              o                   |
        |              o                   |
        |                                  |
```

- In the inheritance model, a base class & data member classes are a subobject of a derived class. So, a resultant memory map looks like the above diagram.
- Virtual table with all overridden virtual functions and code to assign `_vptr` with the virtual table is generated in the constructor of the class by the compiler. I have written more on this topic in the virtual function series.

Coroutine in C Language

Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities

Regex C++

Using std::map Wisely With Modern C++

SCROLL

```cpp

    virtual ~X() {}
    virtual void printX() {}
};

class Y {
public:
    int     y;

    virtual ~Y() {}
    virtual void printY() {}
};

class Z : public X, public Y {
public:
    int     z;

    ~Z() {}
    void printX() {}
    void printY() {}
    void printZ() {}
};
```

- Memory layout:

```
      |                               |
      |-------------------------------| <------ Z class object memory layout
stack |           int X::x            |
  |   |-------------------------------|                  |--------------------------|
  |   |          X:: _vptr            |----------------->|        type_info Z       |
  |   |-------------------------------|                  |--------------------------|
 \|/  |           int Y::y            |                  |    address of Z::~Z()    |
      |-------------------------------|                  |--------------------------|
      |          Y:: _vptr            |------|           |   address of Z::printX() |
      |-------------------------------|      |           |--------------------------|
      |           int Z::z            |      |           |--------GUARD_AREA--------|
      |-------------------------------|      |           |--------------------------|
      |               o               |      |---------->|        type_info Z       |
      |               o               |                  |--------------------------|
      |               o               |                  |    address of Z::~Z()    |
```

Coroutine in C Language

Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities

Regex C++

Using std::map Wisely With Modern C++

SCROLL

```
        |          Y::printY()          |
        |-------------------------------|
        |           Z::~Z()             |
        |-------------------------------|
        |          Z::printX()          |
        |-------------------------------|
        |          Z::printY()          |
        |-------------------------------|
        |          Z::printZ()          |
        |-------------------------------|
        |               o               |
        |               o               |
        |                               |
```

- In multiple inheritance hierarchy, an exact number of the virtual table pointer(`_vptr`) created will be N-1, where N represents the number of classes.
- Now, the rest of the things will be easy to understand for you, I guess so.
- If you try to call a method of class `Z` using any base class pointer, then it will call using the respective virtual table. As an example:

```
 Y *y_ptr = new Z;
 y_ptr->printY(); // OK
 y_ptr->printZ(); // Not OK, as virtual table of class Y doesn't have address of printZ()
method
```

- In the above code, `y_ptr` will point to subobject of class Y within the complete `Z` object.
- As a result, call to any method for say `y_ptr->printY();` using `y_ptr` will be resolved like:

```
   ( *y_ptr->_vtbl[ 2 ] )( y_ptr )
```

- You must be wondering why I have passed `y_ptr` as an argument here. It's because of an implicit `this` pointer, you can learn about it here.

Layout of Object Having Virtual Inheritance

```
 class X { int x; };
 class Y : public virtual X { int y; };
```

Coroutine in C Language

Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities

Regex C++

Using std::map Wisely With Modern C++

SCROLL

```
                    |-----------------|              |------------------|
                    |    Y::_vptr_Y   |------|       |     offset of X  | // offset(20) starts
from Y
  Z class  ------>  |-----------------|      |---->  |------------------|
 sub-object         |     Z::z        |              |      .....       |
                    |-----------------|              |------------------|
                    |    Z::_vptr_Z   |------|
                    |-----------------|      |
  A sub-object -->  |     A::a        |      |       |------------------|
                    |-----------------|      |       |     offset of X  | // offset(12) starts
from Z
  X class ------->  |     X::x        |      |---->  |------------------|
  shared            |-----------------|              |      .....       |
  sub-object        |                 |              |------------------|
```

- Memory representation of derived class having one or more virtual base class divides into two regions:
    1. an invariant region
    2. a shared region
- Data within the invariant region remains at a fixed offset from the start of the object regardless of subsequent derivations.
- However, shared region contains a virtual base class and it fluctuates with subsequent derivation & order of derivation. I have discussed more on this in How Does Virtual Inheritance Works?.

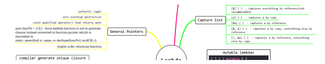
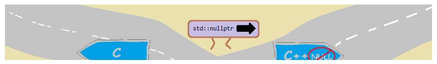
Do you like it ☝️? Get such articles directly into the inbox…!?

Enter your email

FOLLOW

Email Print WhatsApp LinkedIn Reddit Twitter Facebook Messenger

Related Articles

Coroutine in C Language

Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities

Regex C++

Using std::map Wisely With Modern C++

SCROLL

C++20)

the Object Pointed by Base Class Pointer

Coroutine in C Language

Mastering C++: Books | Courses | Tools | Tutorials | Blogs | Communities

Regex C++

Using std::map Wisely With Modern C++

SCROLL