

## 16 POINTERS in C++

原始指针`raw pointer` 智能指针`smart pointer`

编程中最重要的也许就是`memory`（内存），因为我们在代码中所做的每一件事，都是从内存中读取或者写入内存

### 1. What is pointer

指针是一个`integer`，一个整数，它存储一个内存地址

#### 想象

电脑中的内存，就像一条很长的大街，街边有一排房子，每栋房子都有一个门牌号和地址

现在把街上每个带地址的房子想象成一个`byte`，是一个字节的数据。我们需要一个方法来给所有bytes取址  
假设有人在网上订购了一些东西并想要送货上门，它需要被送到正确的房子里，或者有人把东西从它们的房子里送出去。不管怎样，你都需要能够从内存中，那些bytes中，也就是那些房子中`read and write`（读写）

而指针就是那个地址，告诉我们房子在哪儿——那个特定的字节的内存存在哪儿

所以指针就是一个地址，一个存储着一个**内存地址**的整数

`types`(类型)完全没有意义，所有类型的指针都是一个整数，存放一个内存地址

最纯粹的指针 `void` 没有类型，我们给指针定义类型可能代表那个地址的数据是我们给它的类型，让我们的生活从语法上来说更轻松。但是类型并不会改变一个指针，它就只是个整数。

C++

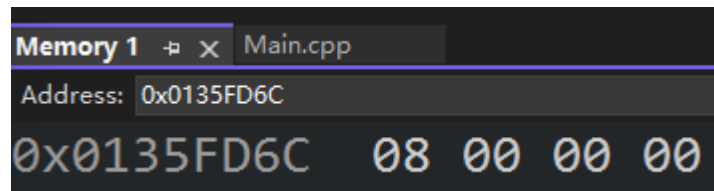
```
void* ptr = 0 ;    // 地址为0无意义，指针处于无效状态，无法访问
void* ptr = NULL ; // (#define NULL 0)
void* prt = nullptr ; // c++11 引入
```

### 2. 指针使用

135fd6c就是内存中存储这个整型变量的位置

```
int var = 8;
void* ptr = &var;
std::cin >> ptr; // 0x0135fd6c
```

可以在VS的memory中查看： [方式请见](#)



Memory 1 Main.cpp  
Address: 0x0135FD6C  
0x0135FD6C 08 00 00 00

可以看到这个地址存储着 8

更改指针类型，依然可以找到内存，类型完全不会造成影响

```
Address: 0x004FFB90
0x004FFB90  08 00 00 00 cc cc cc cc
0x004FFBB6  86 00 f0 13 86 00 18 fc
0x004FFBDC  00 00 00 00 00 00 00 00

Main.cpp
HelloWorld
1 #include <iostream>
2
3 #define LOG(x) std::cout << x << std::endl
4
5 int main()
6 {
7     int var = 8;
8     double* ptr = (double*)&var;
9     std::cin.get();
```

### 3. 访问和改写

我知道数据在哪儿，我怎么访问它呢？

#### Dereferencing(逆向引用):

通过在指针前面加上一个`asterisk(*)`来实现，`*ptr`实际上是在逆向引用这个指针，也就是我现在正在访问的那块数据，我可以读取或者写入那块数据。

```
int var = 8;
void* ptr = var;
*ptr = 10; // 会得到错误，计算机怎么才能将一个值写入void指针里？
```

因为不知道10到底是什么，是一个`short`(两字节的整数)吗，还是一个`int`(四字节整数)，还是一个`long long`(八字节整数)

这时就体现出`type`的重要性了

```
int var = 8;
int* ptr = var;
*ptr = 10;
```

```
Address: 0x0093FBA0
0x0093FBA0  0a 00 0
0x0093FBB1  1f 5d 0
0x0093FBC2  00 00 9

Main.cpp
HelloWorld
7 | int var = 8;
8 | int* ptr = &var;
9 | *ptr = 10;
```

## memset

分配内存，接受指针，数据，存放大小

```
0x016135D8  00 00 00 00 00 00 00 00 00
0x016135E9  d0 14 46 dd 1c 00 80 dd
0x016135FA  dd dd dd dd dd dd dd dd

Main.cpp
HelloWorld (Global Scope)
4
5 int main()
6 {
7   char* buffer = new char[8];
8   memset(buffer, 0, 8);
9   std::cin.get();
```

由于我们使用了`new`这个关键字，所以该数据分配在`heap`(堆)上，因此处理完还需要删除这个数据：

```
char* buffer = new char[8];
memset(buffer, 0, 8); //用于设置一段内存的值。在这里将缓冲区的所有字节都设置为0，实现了清零的效果。

delete[] buffer; //删除数据
```

## 指针的指针

```
Address: 0x003DF80C
0x003DF80C  08 34 96 00 cc cc
0x003DF81D  21 2d 00 01 00 00
0x003DF82E  00 00 80 2a 96 00

Main.cpp
+ HelloWorld (Global Scope)
7 char* buffer = new char[8];
8 memset(buffer, 0, 8);
9
10 char** ptr = &buffer;
11
```

ptr 0x003df80c {0x00963408 ""}

我们知道这个指针将是4字节，因为我们运行的是32位程序（在32位程序里，一个内存地址是32位）这里要注意的是字节序：

### 小端方式

电脑的*endianness is in reverse order*(字节序逆序)，即低位匹配低地址，高位匹配高地址因此图中的地址其实是：00963408

```
Memory 1
Address: 0x00963408
0x00963408  00 00 00 00 00 00 00 00
0x00963419  a9 b9 db dd 14 00 80 dd
```

综上，*pointer*并不是神秘的魔法，也不难，只要明白了它的*gist*，后面的路自然会向你敞开。