**指针教程**

定义：

**指针**是一个变量，其值为另一个变量的地址，即，内存位置的直接地址。就像其他变量或常量一样，您必须在使用指针存储其他变量地址之前，对其进行声明。

int \*ip; /\* 一个整型的指针 \*/

double \*dp; /\* 一个 double 型的指针 \*/

float \*fp; /\* 一个浮点型的指针 \*/

char \*ch; /\* 一个字符型的指针 \*/

1、指针的简单使用：

指向基本数据类型的使用

#include <iostream>

using namespace std;

int main ()

{

int var = 20; // 实际变量的声明

int \*ip; // 指针变量的声明

ip = &var; // 在指针变量中存储 var 的地址，注意指针一定要指向变量的地址

cout << "Value of var variable: ";

cout << var << endl; // 输出在变量的值

cout << "Address stored in ip variable: ";

cout << ip << endl; // 访问指针中地址，其实就是var变量在内存中的地址

cout << "Value of \*ip variable: ";

cout << \*ip << endl; //访问指针所指内存地址里存储的值。其实就是var值

return 0;

}

指向类的指针的使用

#include <iostream>

using namespace std;

class A

{

private:

int value;

public:

int k;

A()

{

value = 10;

k = 7;

}

void print()

{

cout<<value<<endl;

}

};

int main ()

{

A Avar; // 实际变量的声明

A \*p = &Avar;

注意观察两者的区别

Avar.print(); //变量调用函数

p->print(); //指向类的指针调用函数

cout<<p->k<<endl; //指针调用类中public变量。

return 0;

}

2、指针与数组

指针和数组是密切相关的。事实上，指针和数组在很多情况下是可以互换的。一个指向数组开头的指针，可以通过使用指针的算术运算或数组索引来访问数组。

#include <iostream>

using namespace std;

const int MAX = 3;

int main ()

{

int var[MAX] = {10, 100, 200};

int \*ptr; // 指针中的数组地址

ptr = var; //指针可以直接指向数组名称，var前面一定不能加&，

//而且指针的类型一定要与数组的类型一致。

for (int i = 0; i < MAX; i++)

{

/**/当指针指向数组的时候，大多数情况下指针可以当成数组用**

cout << "var[" << i << "]的内存地址为 ";

cout << &ptr[i] << endl; //输出数组var的第i个数所在的内存地址

cout << "var[" << i << "] 的值为 ";

cout << ptr[i] << endl; //输出数组var的第i个数的值

}

return 0;

}

**3、C++ 指针的算术运算**

指针是一个用数值表示的地址。因此，您可以对指针执行算术运算。可以对指针进行四种算术运算：++、--、+、-。 假设 ptr 是一个指向地址 1000 的整型指针，是一个 32 位的整数，让我们对该指针执行下列的算术运算：

ptr++

在执行完上述的运算之后，**ptr** 将指向位置 1004，因为 ptr 每增加一次，它都将指向下一个整数位置，即当前位置往后移 4 个字节。这个运算会在不影响内存位置中实际值的情况下，移动指针到下一个内存位置。如果 **ptr** 指向一个地址为 1000 的字符，上面的运算会导致指针指向位置 1001，因为下一个字符位置是在 1001。

补充：指针做运算时，位置移动的长度与声明指针的类型长度一致。声明整型指针，那么做++后，地址会增加4个单位的长度（整型的长度是4）.声明字符指针，那么做++后，地址增加1（字符的长度是1）。

**4、指针与动态数组用法（重点）**

在做类似“学生管理系统”中，很多人喜欢定义一个全局数组变量student st[500]存放学生数据。但是随着学生人数增多，500个长度的数组一定不够用。若要扩充学生数据，那么可以采用动态生成数组的方式进行扩充。由第二点可知：只要指针指向数组，那么大多数指针可以当成数组使用。因此不能声明全局数组，而是声明全局指针。其次，需要用到关键字new。具体用法如下：

#include <iostream>

using namespace std;

int main ()

{

//背景：在长度为3的数组上多加1个数，数值为15

int Avar[3] = {2,5,8}; //要扩充的数组

int \*p = new int[4]; //new的意思是向内存申请长度为4个int的数组空间

//让p指向申请的数组空间

//此时p相当于指向一个长度为4的int数组

int i = 0;

for(i=0;i<3;i++) //对p中前3个数赋值

{

p[i] = Avar[i];

}

p[3] = 15; //对p中的第4个数赋值

for(i=0;i<4;i++) //此时p所指向的数组已经存放Avar的数据以及第4个数据

{

cout<<p[i]<<endl;

}

delete []p; //使用结束后要删除指针，否则出现内存泄漏（重要）

p = NULL; //一定要将指针置空，否则该指针有效（重要）

return 0;

}

**5、指针引用**

思考：下列代码中swap1 和swap2看起来类似，为什么调用swap1后，a、b的值会交换，但是调用swap2后，a、b的值不会交换？

#include <iostream>

using namespace std;

void swap1(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void swap2(int a, int b)

{

int temp = a;

a = b;

b = temp;

}

void swap3(int &a, int &b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main ()

{

int a = 10;

int b = 20;

cout<<"a = "<<a<<"\t"<<"b = "<<b<<endl;

//输出结果是：a = 10 b = 20

swap1(&a,&b);

cout<<"a = "<<a<<"\t"<<"b = "<<b<<endl;

//输出结果是：a = 20 b = 10

Swap2(a,b);

cout<<"a = "<<a<<"\t"<<"b = "<<b<<endl;

//输出结果是：a = 10 b = 20

Swap3(a,b);

cout<<"a = "<<a<<"\t"<<"b = "<<b<<endl;

//输出结果是：a = 20 b = 10

}

在C++中，[参数传递](https://so.csdn.net/so/search?q=%E5%8F%82%E6%95%B0%E4%BC%A0%E9%80%92&spm=1001.2101.3001.7020" \t "_blank)的方式是“实虚结合”。主要有三种方式

* 按值传递(pass by value)
* 地址传递(pass by pointer)
* 引用传递(pass by reference)

按[值传递](https://so.csdn.net/so/search?q=%E5%80%BC%E4%BC%A0%E9%80%92&spm=1001.2101.3001.7020)的过程为：首先计算出实参表达式的值，接着给对应的形参变量分配一个存储空间，该空间的大小等于该形参类型的，然后把以求出的实参表达式的值一一存入到形参变量分配的存储空间中，成为形参变量的初值，供被调用函数执行时使用。这种传递是把实参表达式的值传送给对应的形参变量，故称这种传递方式为“按值传递”。

使用这种方式，调用函数本身不对实参进行操作，也就是说，即使形参的值在函数中发生了变化，实参的值也完全不会受到影响，仍为调用前的值。

void swap2(int a, int b)函数属于值传递，因此调用函数后，a、b值不变。

值传递的形参变量将会在函数结束后销毁（重点）

地址传递如果在函数定义时将形参说明成指针，对这样的函数进行调用时就需要指定地址值形式的实参。这时的参数传递方式就是地址传递方式。

地址传递与按值传递的不同在于，它把实参的存储地址传送给对应的形参，从而使得形参指针和实参指针指向同一个地址。因此，被调用函数中对形参指针所指向的地址中内容的任何改变都会影响到实参。

void swap1(int \*a, int \*b)属于地址传递，调用函数后，a、b值会互换。

引用传递：如果以引用为参数，则既可以使得对形参的任何操作都能改变相应的数据，又使得函数调用显得方便、自然。[引用传递](https://so.csdn.net/so/search?q=%E5%BC%95%E7%94%A8%E4%BC%A0%E9%80%92&spm=1001.2101.3001.7020)方式是在函数定义时在形参前面加上引用运算符“&”。

void swap3(int &a, int &b)属于引用传递，调用函数后，a、b值会互换。

第4种方式：采用关键字const

**const关键字。**当你的参数是作为输入参数时，你总不希望你的输入参数被修改，否则有可能产生逻辑错误，这时可以在声明函数时在参数前加上const关键字，防止在实现时意外修改函数输入，对于使用你的代码的程序员也可以告诉他们这个参数是输入，而不加const关键字的参数也可能是输出。例如strlen，你可以这样声明

     int strlen(char str)

 功能上肯定没有什么问题，但是你想告诉使用该函数的人，参数str是一个输入参数，它指向的数据是不能被修改的，这也是他们期望的，总不会有人希望在请人给他数钱的时候，里面有张100的变成10块的了，或者真钞变成假钞了，他们希望有一个保证，说该函数不会破坏你的任何数据，声明按如下方式便可让他们放心：

     int strlen(const char str)

采用const关键字，不会对原数据做任何修改，也不会在函数调用时会给参数分配一个空间。