喜欢专研自己所热爱的知识，以及项目完成的那一份喜悦感  
加油！当自己的实力不能满足自己的目标时，就**静下心去学习**！

@[toc]

# 指针

## 指针的基本概念

**指针的作用：** 可以通过指针间接访问内存

* 内存编号是从0开始记录的，一般用十六进制数字表示
* 可以利用指针变量保存地址

## 指针变量的定义和使用

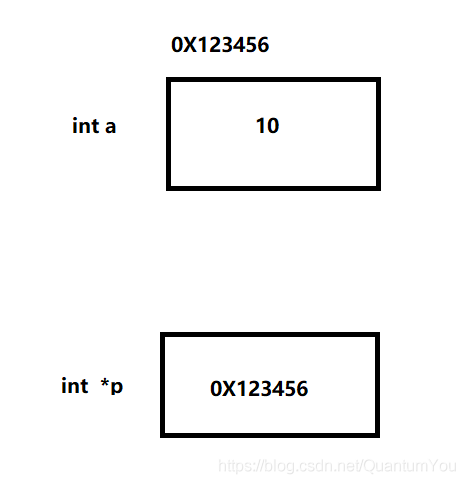
* 指针变量定义语法： 数据类型 \* 变量名；

**示例：**

int main() {  
  
 //1、指针的定义  
 int a = 10; //定义整型变量a  
   
 //指针定义语法： 数据类型 \* 变量名 ;  
 int \* p;  
  
 //指针变量赋值  
 p = &a; //指针指向变量a的地址  
 cout << &a << endl; //打印数据a的地址  
 cout << p << endl; //打印指针变量p  
  
 //2、指针的使用  
 //通过\*操作指针变量指向的内存  
 cout << "\*p = " << \*p << endl;  
  
 system("pause");  
  
 return 0;  
}

指针变量和普通变量的区别

* 普通变量存放的是数据,指针变量存放的是地址
* 指针变量可以通过" \* "操作符，操作指针变量指向的内存空间，这个过程称为解引用



总结 :

* 1、我们可以通过 & 符号 获取变量的地址
* 2、利用指针可以记录地址
* 3、对指针变量解引用，可以操作指针指向的内存

## 指针所占内存空间

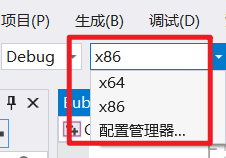
* 指针也是种数据类型，那么这种数据类型占用多少内存空间？

**示例：**

int main() {  
  
 int a = 10;  
  
 int \* p;  
 p = &a; //指针指向数据a的地址  
  
 cout << \*p << endl; //\* 解引用  
 cout << sizeof(p) << endl;  
 cout << sizeof(char \*) << endl;  
 cout << sizeof(float \*) << endl;  
 cout << sizeof(double \*) << endl;  
  
 system("pause");  
  
 return 0;  
}

**总结**：所有指针类型在32位操作系统下是4个字节，在64位操作系统下是8个字节

* x86\_32 简写为x86
* x86\_64 --> x64



## 空指针与野指针

* **空指针**：指针变量指向内存中编号为0的空间
* **用途** ： 初始化指针变量

**注意**：空指针指向的内存是不可以访问的

* 内存编号0 ~255为系统占用内存，不允许用户访问

**示例1：空指针**

int main() {  
  
 //指针变量p指向内存地址编号为0的空间  
 int \* p = NULL;  
  
 //访问空指针报错   
 //内存编号0 ~255为系统占用内存，不允许用户访问  
 cout << \*p << endl;  
  
 system("pause");  
  
 return 0;  
}

* **野指针**：指针变量指向非法的内存空间

**示例2：野指针**

int main() {  
  
 //指针变量p指向内存地址编号为0x1100的空间  
 int \* p = (int \*)0x1100;  
  
 //访问野指针报错   
 cout << \*p << endl;  
  
 system("pause");  
  
 return 0;  
}

总结：空指针和野指针都不是我们申请的空间，因此不要访问。

空指针拓展知识：

* void\* 这不叫空指针,这叫**无确切类型指针**.这个指针指向一块内存,却没有告诉程序该用何种方式来解释这片内存.所以这种类型的指针不能直接进行取内容的操作.必须先转成别的类型的指针才可以把内容解释出来.
* 还有'\0',这也不是空指针所指的内容. '\0'是表示一个字符串的结尾而已,并不是NULL的意思.
* 真正的空指针是说,这个指针没有指向一块有意义的内存,比如说:char\* k; 这里这个k就叫空指针.我们并未让它指向任意地点. 又或者 char\* k = NULL; 这里这个k也叫空指针,因为它指向NULL 也就是0,注意是整数0,不是'\0'
* 一个空指针我们也无法对它进行取内容操作.空指针只有在真正指向了一块有意义的内存后,我们才能对它取内容.也就是说要这样 k = "hello  
  world!"; 这时k就不是空指针了.

## const修饰指针

const修饰指针有三种情况

1. const修饰指针 --- 常量指针
2. const修饰常量 --- 指针常量
3. const即修饰指针，又修饰常量

[解释视频 1:00](https://www.bilibili.com/video/BV1et411b73Z?p=60&spm_id_from=pageDriver)

**示例：**

int main() {  
  
 int a = 10;  
 int b = 10;  
  
 //const修饰的是指针，指针指向可以改，指针指向的值不可以更改  
 const int \* p1 = &a;   
 p1 = &b; //正确  
 //\*p1 = 100; 报错  
   
  
 //const修饰的是常量，指针指向不可以改，指针指向的值可以更改  
 int \* const p2 = &a;  
 //p2 = &b; //错误  
 \*p2 = 100; //正确  
  
 //const既修饰指针又修饰常量  
 const int \* const p3 = &a;  
 //p3 = &b; //错误  
 //\*p3 = 100; //错误  
  
 system("pause");  
  
 return 0;  
}

技巧：看const右侧紧跟着的是指针还是常量, 是指针就是常量指针，是常量就是指针常量

## 指针和数组

* **作用** ： 利用指针访问数组中元素

**示例：**

int main() {  
  
 int arr[] = { 1,2,3,4,5,6,7,8,9,10 };  
  
 int \* p = arr; //指向数组的指针  
  
 cout << "第一个元素： " << arr[0] << endl;  
 cout << "指针访问第一个元素： " << \*p << endl;  
  
 for (int i = 0; i < 10; i++)  
 {  
 //利用指针遍历数组  
 cout << \*p << endl;  
 p++;  
 }  
  
 system("pause");  
  
 return 0;  
}

## 指针和函数

* **作用** : 利用指针作函数参数，可以修改实参的值

**示例：**

//值传递  
void swap1(int a ,int b)  
{  
 int temp = a;  
 a = b;   
 b = temp;  
}  
//地址传递  
void swap2(int \* p1, int \*p2)  
{  
 int temp = \*p1;  
 \*p1 = \*p2;  
 \*p2 = temp;  
}  
  
int main() {  
  
 int a = 10;  
 int b = 20;  
 swap1(a, b); // 值传递不会改变实参  
  
 swap2(&a, &b); //地址传递会改变实参  
  
 cout << "a = " << a << endl;  
  
 cout << "b = " << b << endl;  
  
 system("pause");  
  
 return 0;  
}

**总结**：如果不想修改实参，就用值传递，如果想修改实参，就用地址传递

[解释视频 6：20](https://www.bilibili.com/video/BV1et411b73Z?p=62&spm_id_from=pageDriver)

## 总结案例

* 利用指针、数组、函数 的知识,封装一个函数，利用冒泡排序，实现对整型数组的升序排序

例如数组：int arr[10] = { 4,3,6,9,1,2,10,8,7,5 };

//冒泡排序函数  
void bubbleSort(int \* arr, int len) //int \* arr 也可以写为int arr[]  
{  
 for (int i = 0; i < len - 1; i++)  
 {  
 for (int j = 0; j < len - 1 - i; j++)  
 {  
 if (arr[j] > arr[j + 1])  
 {  
 int temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 }  
 }  
 }  
}  
  
//打印数组函数  
void printArray(int arr[], int len)  
{  
 for (int i = 0; i < len; i++)  
 {  
 cout << arr[i] << endl;  
 }  
}  
  
int main() {  
  
 int arr[10] = { 4,3,6,9,1,2,10,8,7,5 };  
 int len = sizeof(arr) / sizeof(int);  
  
 bubbleSort(arr, len);  
  
 printArray(arr, len);  
  
 system("pause");  
  
 return 0;  
}