[指针的使用](#header-n2)  
 [1 什么是指针](#header-n3)  
 [2 如何定义指针](#header-n6)  
 [3 指针变量的基本使用](#header-n17)  
 [4 指针的存储](#header-n25)  
 [5 指针的初始化](#header-n33)  
 [6 指针所支持的运算](#header-n41)  
 [7 多重指针](#header-n82)  
 [8 指针与一维数组](#header-n102)  
 [9 指针与字符串](#header-n133)  
 [10 几种特殊的指针](#header-n163)  
 [11 指针与二维数组](#header-n182)

### 指针的使用

#### 1 什么是指针

指针就是地址！

指针变量是用来存储地址的！

**特殊之处：快，方便有序的安排地址……**

#### 2 如何定义指针

int num = 100;  
int \* p = &num;

* 定义了一个存储有整型变量num地址的指针变量p
* p可以称为整型指针
* 因为p存储的是num的地址，通常称p指向num
* p这个指针变量的类型是 int \*
* **指针类型是指除去变量名后的部分，指针类型决定**
* **指针直指向的类型指的是去除\*和变量名后的内容指针指向的类型决定索取内容空间的大小，和指针+1跳过的单位长度**
* **p = xxx 指的是将xxx对应的空间的值赋值给p指向的地址空间，而不是赋值给p的地址空间**

#### 3 指针变量的基本使用

* & ： 取地址符，获取一个变量的地址
* \* ： 寻址符，通过一个指针（地址）找到它所对应的内存空间

#include <stdio.h>  
  
int main(void)  
{  
 int num = 100;  
 // 定义了一个存储有整型变量地址的指针  
 // 并将num的地址赋值给他  
 int \* p = &num;  
 // p中存储了num的地址，我们就说p指向num  
  
 \*p = 200;  
  
 printf("num=%d\n", num);  
 printf("num=%d\n", \*p);  
  
 printf("%p\n", &num);//%p 输出指针的值，地址值  
 printf("%p\n", p);  
  
 return 0;  
}

int num = 100;  
int \* p = &num;  
   
printf("&num = %p\n", &num);// 打印num的地址  
printf("p = %p\n", p); // 打印p的值 p中存储的就是num的地址  
printf("num = %d\n", num); // 打印num的值  
printf("\*p = %d\n", \*p); // 打印\*p的值 等同于 num的值  
printf("&p = %p\n", &p); // 打印指针变量p的地址  
  
printf("%d\n", sizeof(p)); // 求的是指针变量p所占用的内存空间  
printf("%d\n", sizeof(\*p)); // 求的是num所占用的内存空间  
printf("%d\n", sizeof(num));// 求的是num所占用的内存空间

#### 4 指针的存储

* 32位机，指针占用4个字节内存空间
* 64位机，指针占用8个字节内存空间
* 但是通常开发环境中无论32位机还是64位机，指针都占4个字节

#### 5 指针的初始化

* **没有进行初始化的指针称为 野指针，指向哪是不确定的**
* 对于指针初始化可以赋值为某个变量的地址
* 如果指针在定义时指向哪不确定，需要在程序运行过程中进行赋值，那么它的**初始值可以赋值为 NULL**，**称这种指针为空指针**

#### 6 指针所支持的运算

* \* : 寻址符，根据指针存储的地址找到对应的内存，**寻址的结果是一块内存空间，**可进行写入或读取操作
* &：取地址符，获取某个变量的地址（地址就是指针，变量的地址也是附带有类型信息的）
  + int num=3; &num type: int \*
  + double pi=3.14; &pi type: double \*
* = ：赋值运算符，可以将一个地址赋值给指针变量
* ==、!=、>、< ： 关系运算符
  + ==、!= 是最常用的，表示判断两个指针指向的**是否是同一块内**存
  + > 、< 不太常用，判断的就是指针存储地址的大小关系
* +：指针 + 整数
  + 指针在加运算中，每加一，实际上是往后移动一个所指向变量类型单位的长度
  + int num=10; int \* p=&num; p + 1 相当于往**后移动一个int单位**
* -：指针 - 整数
  + 指针在减运算中，每减一，实际上是往**前**移动一个所指向变量类型单位的长度
* -： 指针 - 指针
  + 指针-指针 求的是两个指针地址之间，单位元素的个数（单位就是指针指向的变量的类型）
  + int arr[4] = {1,2,3,4};  
    int \* p1 = &amp;arr[0];  
    int \* p2 = &amp;arr[3];  
    printf(&quot;%d\n&quot;, p2 - p1); // 3
  + 不存在 指针 + 指针 运算！

#### 7 多重指针

* 若一个指针存储的内容是另一个指针的地址，则称这个指针为多重指针。
* 多重指针就是指向指针的指针
* 二级指针：就是指向一级指针的指针
* 三级指针：就是指向二级指针的指针
* 注意：
  + 允许连续**寻址**
  + 不允许连续**取地址**
  + 在使用中，一定要注意多重指针的等价表示形式

#include <stdio.h>  
  
int main(void)  
{   
 int num = 10;  
 int \* p1 = &num; // &num -> 指针（地址），带类型 int \*  
 int \*\* p2 = &p1; // p2 二级指针，指向一级指针的指针  
   
 printf("num = %d\n", num); // num这块内存，10  
 printf("\*p1 = %d\n", \*p1); // num这块内存，10  
 printf("\*\*p2 = %d\n", \*\*p2); // num这块内存，10  
  
 printf("&num = %p\n", &num); // p1这块内存，num的地址  
 printf("p1 = %p\n", p1); // p1这块内存，num的地址  
 printf("\*p2 = %p\n", \*p2); // p1这块内存，num的地址  
  
 printf("&p1 = %p\n", &p1); // p1的地址  
 printf("p2 = %p\n", p2); // p1的地址  
  
 printf("&&num = %p\n", &&num); // 错误，不允许连续取地址  
 // 因为&num获取的仅仅是一个值  
 // 不能对一个值进行取地址  
 // 只能对一块内存进行取地址  
  
 return 0;  
}

#### 8 指针与一维数组

* 当一个指针指向数组某元素时，可以通过这个指针加减来访问数组中的每一个元素
* 下标访问元素 本质上就是 指针加减寻址

#include <stdio.h>  
  
int main(void)  
{   
 int i;  
 int arr[5] = {11, 22, 33, 44, 55};  
 int \* p = &arr[0];  
  
 for (i=0; i<5; i++) {  
 // 下标访问元素 和 指针寻址访问元素 本质上是等价的  
 // arr[i] => \*(arr+i) 下标本质上就是指针寻址  
 printf("%d[arr] = %d\n", i, i[arr]);  
 printf("arr[%d] = %d\n", i, arr[i]);  
 printf("\*(arr+%d) = %d\n", i, \*(arr+i));  
 printf("\*(p+%d) = %d\n", i, \*(p+i));  
 printf("p[%d] = %d\n", i, p[i]);  
 }  
  
 return 0;  
}

* arr 在数值层面 它表示的就是 arr数组首元素的地址（指针）
* arr 在变量层面 它表示的就是 一个长度为5元素类型为int的数组
* 数组名，可以理解成它是一个特殊的指针，类似于指针而又跟指针有区别

#include <stdio.h>  
  
int main(void)  
{   
 int i;  
 int arr[5] = {11, 22, 33, 44, 55};  
 int \* p = arr;

printf("%d\n", \*p); // &arr[0] => &\*(arr+0) => arr+0 => arr   
  
 return 0;  
}

* 指针与数组名的相同点和不同点

#include <stdio.h>  
  
int main(void)  
{   
 int i;  
 int arr[5] = {11, 22, 33, 44, 55};  
 int \* p = arr;   
  
 // 相同点  
 for (i=0; i<5; i++) {  
 printf("%d\n", arr[i]); // \*(arr+i)  
 printf("%d\n", p[i]); // \*(p+i)  
 }  
  
 // 不同点  
 printf("%p\n", arr); // 数组首元素（arr[0]）的地址  
 printf("%p\n", p); // 数组首元素（arr[0]）的地址  
 printf("%p\n", &arr); // 整个数组（int arr[5]）的地址  
 printf("%p\n", &p); // 指针变量p的地址  
  
 printf("arr + 1 = %p\n", arr + 1); // 移动一个int单位  
 printf("&arr + 1 = %p\n", &arr + 1); // 移动一个数组（int [5]）单位  
  
 printf("sizeof(arr) = %d\n", sizeof(arr)); // 20 求的是整个数组（in t arr[5]）的大小  
 printf("sizeof(p) = %d\n", sizeof(p)); // 4 求的是指针变量p的 内存大小  
  
 return 0;  
}

* 数组指针
  + 本质上是一个指针，指向数组的指针
* 指针数组
  + 本质上是一个数组，存储指针的数组

int arr[5] = {11, 22, 33, 44, 55};  
// p 就是数组指针  
int (\* p)[5] = &arr;   
// brr 就是一个指针数组，每一个元素都是一个指针  
int \* brr[5];

#### 9 指针与字符串

* 字符串
  + 存储有字符串的字符数组
  + 字符串字面值
    - "ABCD" 本质上是一个指针，指针指向内存的只读区，对应的只读区地址对应内存保存的就是 A B C D \0
    - 字符串字面值，实际上会在只读区存储一个字符数组，数组中的元素就是字符串字面值里面的字符+\0，然后字符串字面值就是指向这个数组首元素的地址。

#include <stdio.h>  
  
int main(void)  
{   
 char arr[] = "ABCD"; // arr是一个字符数组，数组里面保存着 ABCD\0  
 // 字符串的数据保存在自己的内存中  
 char \* p = "ABCD"; // p是一个字符指针，指向只读区保存 ABCD\0 的位置  
 // 字符串的数据保存在只读区  
  
 printf("%s\n", arr); // 传入的是arr数组首元素的地址  
 printf("%s\n", p); // 传入的是指针p保存的地址，只读区存放ABCD\0数组首元素地址  
 printf("%s\n", "ABCD"); // 只读区存放ABCD\0数组首元素地址  
  
  
 arr[0] = 'B'; // 修改数组arr首元素的值，允许！栈区  
 // \*p = 'B'; // 尝试往只读区写入数据，不允许！  
  
 // arr = "1234"; // arr不允许指向其它位置  
 p = "1234"; // p指向只读区存放 1234\0的位置  
  
 return 0;  
}

* 字符串在输出时，需要传入一个指针（地址），然后从这个地址开始依次取字符，如果当前字符不是'\0'则输出，并且指针+1，继续上述操作，指导遇到'\0'为止。

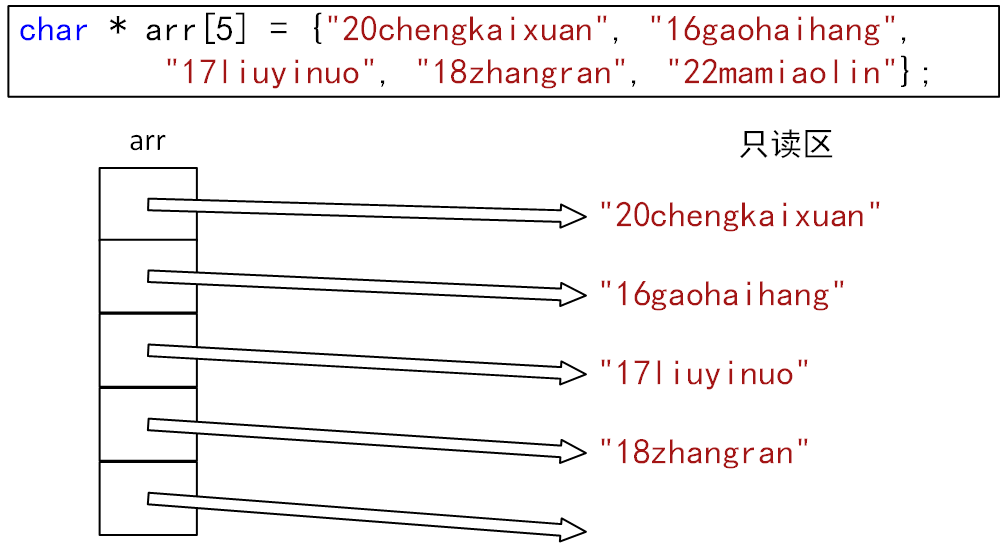
#include <stdio.h>  
  
int main(void)  
{   
 char arr[] = "ABCD1234";   
 char \* p = "ABCD1234";   
  
 // 字符串输出函数需要是一个指向存储有字符串内容  
 // 的字符数组首元素的地址  
 printf("%s\n", arr+4);  
 printf("%s\n", p+4);  
 printf("%s\n", "ABCD1234"+4);  
  
 printf("%s\n", arr);  
 printf("%s\n", p);  
 printf("%s\n", "ABCD1234");  
  
 return 0;  
}

strcmp(); // 两个参数必须都能读  
strlen(); // 参数必须能读  
strcat(); // 参数1必须能写（不能是字符串字面值），参数2必须能读  
strcpy(); // 参数1必须能写（不能是字符串字面值），参数2必须能读

* 字符指针数组

// 字符指针数组  
// 数组里面每一个元素都是一个字符指针  
char \* arr1[5] = {"111", "222", "333", "444", "555"};  
char \* temp;  
int i;  
for (i=0; i<5; i++) {  
 puts(arr1[i]);  
}  
  
temp = arr1[1];  
arr1[1] = arr1[2];  
arr1[2] = temp;  
  
puts("=================");  
for (i=0; i<5; i++) {  
 puts(arr1[i]);  
}

* 字符指针数组保存字符串，排序



#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{   
 char \* arr[5] = {"20chengkaixuan", "16gaohaihang",   
 "17liuyinuo", "18zhangran", "22mamiaolin"};  
 int i, j, len;  
 len = 5;  
 for (i=0; i<len-1; i++) {  
 for (j=0; j<len-1-i; j++) {  
 if (strcmp(arr[j], arr[j+1]) > 0) {  
 char \* t = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1] = t;  
 }  
 }  
 }  
  
 for (i=0; i<5; i++)  
 puts(arr[i]);  
  
 return 0;  
}

#### 10 几种特殊的指针

* const 指针
  + 在指针赋值的时候，只允许加限制，不允许减限制

const int num = 10;  
// 丢常，在赋值的过程中丢失了const的限制，会报错  
int \* p = &num; // const int \* -> int \*  
\*p = 100;

#include <stdio.h>  
  
int main(void)  
{   
 int num = 10;  
 // 常量指针，指向常量的指针  
 const int \* p1 = &num; // 表示 \*p1 不允许修改  
 int const \* p2 = &num; // 表示 \*p2 不允许修改  
 // 指针常量，指针本身是常量  
 int \* const p3 = &num; // 表示 p3 不允许被修改  
 // 常量常指针  
 int const \* const p4 = &num; // 表示p4 不允许被修改，\*p4也不允许被修改  
  
 return 0;  
}

#### 11 指针与二维数组

* 指针与一维数组相关知识
  + **数组名**在数值计算中表示**数组首元素的地址**
  + 数组名跟指针很相似，但又不同
* 二维数组的相关知识
  + 二维数组本质上是**元素类型是一维数组的一维数组**。
* **数组的地址进行寻址，得到的是数组首元素的地址**

int brr[4] = {11, 12, 13, 14};  
  
 // &brr : 长度为4的int一维数组的地址  
 // \*&brr : brr, 长度为4的int一维数组的首元素地址  
 // 数组的地址进行寻址，得到的是数组首元素的地址

#include <stdio.h>  
  
int main(void)  
{  
  
  
 // 1、将二维数组抽象理解成一维数组  
 // arr可以理解成是一个长度为3的一维数组  
 // 每个元素是长度为4的int一维数组  
 // 2、根据一维数组理解二维数组数组名的含义  
 // arr 在数值计算中表示的含义是什么？  
 // arr表示长度为3的一维数组（二维数组arr）的首元素的地址  
 // 首元素就是一个长度为4的int一维数组  
 // 首元素的地址就是第一个长度为4的int一维数组的地址  
 // arr表示的就是 第一个长度为4的int一维数组的地址（第一行的地址）  
 // 3、解释二维数组名寻址之后表示的含义  
 // \*arr是什么？  
 // \*arr表示：第一个长度为4的int一维数组的首元素的地址（11的地址）  
 // 4、解释二维数组名寻址之后再寻址所表示的含义  
 // \*\*arr是什么？  
 // \*\*arr表示：第一个长度为4的int一维数组的首元素（11）  
 // 5、寻址法 和 下标法 访问元素，下标访问元素本质上就是寻址  
 int arr[3][4] = {{11,12,13,14},  
 {21,22,23,24},  
 {31,32,33,34}};  
 printf("%d\n", \*\*arr); printf("%d\n", arr[0][0]);  
 // arr[0][0] => \*(arr[0]+0) => \*(\*(arr+0)+0) => \*\*arr  
 printf("%d\n", \*(\*arr+1)); printf("%d\n", arr[0][1]);  
 // arr[0][1] => \*(arr[0]+1) => \*(\*(arr+0)+1) => \*(\*arr+1)  
 printf("%d\n", \*\*(arr+1)); printf("%d\n", arr[1][0]);  
 // arr[1][0] => \*(\*(arr+1)+0) => \*\*(arr+1)  
 printf("%d\n", \*(\*(arr+2)+2)); printf("%d\n", arr[2][2]);  
 // arr[2][2] => \*(\*(arr+2)+2)  
  
   
 return 0;  
}

已知：int a[2][3];下列各表达式的含义：  
&a : 二维数组a的地址  
a : 第一行的地址  
&a[0] : 第一行的地址  
a[0] : 第一行第一个元素的地址  
a[0][0] : 第一行第一个元素 \*\*arr  
&a[0][0] : 第一行第一个元素的地址 \*arr

已知：int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};  
则：  
a[1]+2 : 13的地址，a[1]+2 => &\*(a[1]+2) => &a[1][2]  
\*(a+1)+2 : 13的地址，\*(a+1)+2 => a[1]+2  
\*(a[1]+2) : 13, \*(a[1]+2) => a[1][2]  
\*(\*(a+1)+2) : 13, \*(\*(a+1)+2) => a[1][2]  
(\*(a+1))[2] : 13, (\*(a+1))[2] => \*(\*(a+1)+2) => a[1][2]  
以上表达上表示什么？