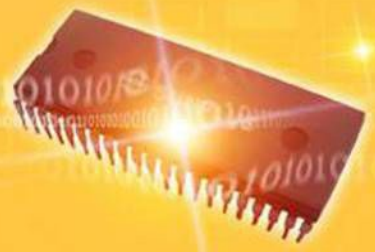
凌阳教育
www.sunplusedu.com
值得信赖的教育品牌

嵌入式系统工程师



Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

指针的概念与应用

指针

爱它，就让它去学C语言的指针

恨它，也让它去学C语言的指针

指针是C语言里面**最重要**也是**最难理解**的知识

学习了指针，才算真正踏入C语言的大门

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
- 函数与指针
- 其它特殊指针

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
  - 存放指针的数组——指针数组
  - 指向数组元素的指针——数组元素指针
  - 指向数组的指针——数组指针（行指针）
- 函数与指针
- 其它特殊指针

存放指针的数组——指针数组

- 如果我们需要一个指针变量，定义如下：

```
char *p;
```

- 如果我们需要10个指针变量，定义如下：

```
char *p1,*p2,*p3;
```

- 这样使用起来很麻烦，那么我们可以将**需要的指针放在一个数组**中——指针数组（存放指针的数组）

```
char  *  p[10];
float *  p[10];
int *    q[4][3];
```

存放指针的数组—指针数组

- 一个数组，即其元素均为指针类型的数据——**指针数组**

  一维指针数组:　　类型名　数组名[数组长度];

  二维指针数组:　　类型名　数组名[行][列];

- 例如

```
int  array[8]= { 1,2,3,4,5,6,7,8 };
```

- 一维数值指针数组:

```
int *p[6]={ &array[0],&array[1],&array[2],&array[3]};
p[4] = &array[4];
p[5] = &array[5];
```

- 二维数值指针数组:

```
int *a[2][3];
a[0][1]=&array[0];
```

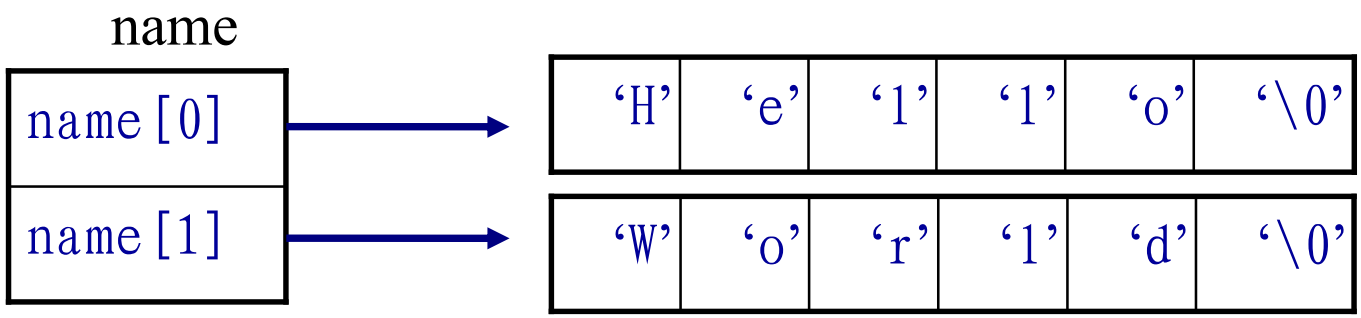
指针数组

- 指针数组与字符串的联系：

```
char *name[2]={ “Hello” , “World” };
name[0] = “Hello” ;
name[1] = “World” ;
```

注意：

不要以为数组中存放的是字符串，它存放的是字符串首地址.这一点一定要注意！



指针数组

- 05.aver_point.c

```
#include<stdio.h>
//指针数组
int main()
{
    char *name[] = { "Follow me","BASIC","GreatWall","FORTRAN","Computer design"};
    int i;
    for( i = 0 ; i<5 ;i++)
    {
        printf("%s\n",name[i]);
    }
    return 0;
}
```

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
  - 存放指针的数组——指针数组
  - 指向数组元素的指针——数组元素指针
  - 指向数组的指针——数组指针（行指针）
- 函数与指针
- 其它特殊指针

指向数组元素的指针

- 数组元素在内存中分配的地址称为数组元素的指针
- 假设：有int array[4]={1,2,3,4};

  经过前面学习我们知道：**每个变量都是有自己的地址**

  同时数组是由相同类型的变量组成的，那么每个数组元素都会有自己的地址
- 那么我们可以定义一个指向数组元素的指针

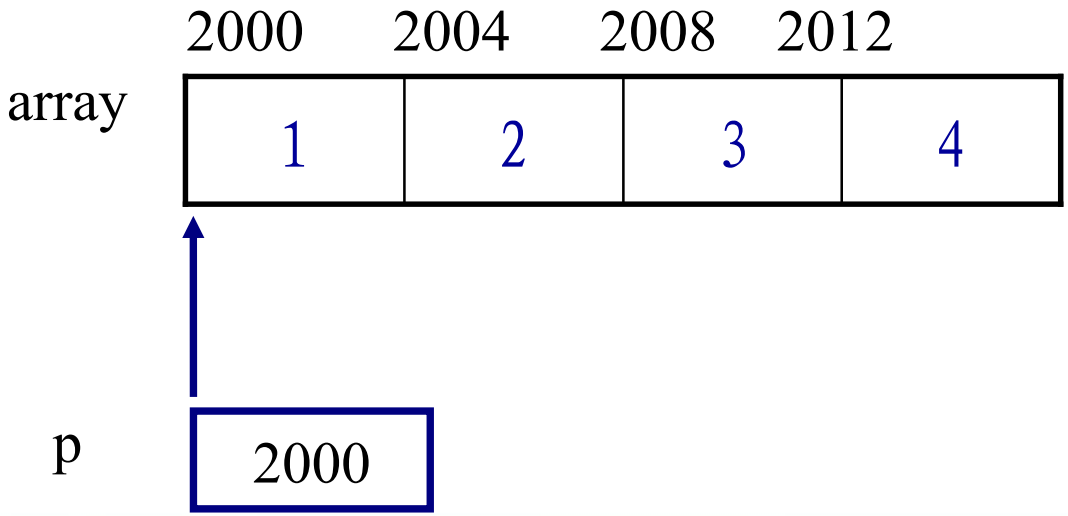
```
int array_1[10];
int array_2[5][3];
int *p,*q;
p = &array_1[0];      //一维数组元素的指针
q = &array_2[2][1];   //二位数组元素的指针
```

跟我们之前讲的指向变量的指针定义方法是一样的

指向数组元素的指针

- 在C语言中，设定一维数组名代表了数组第一个元素的地址，那么以下两种写法是等价的：

```
int array[5];
int *p;
p = &array[0]
p = array;
```



1维数组与指针

- 引用数组元素

1. 下标法

```
array[1];
```

2. 指针法

```
*( p + i );或 *( array + i );
```

注意:

- array为数组的首地址是一个常量
  因此不能进行array++或者++array操作.
- p是变量,其值为array数组的首地址，可以进行++操作.

1维数组与指针

➢ 数组元素的指针：06.aver_point2.c

```
1  #include <stdio.h>
2  //采用指针法引用数组元素
3  int main()
4  {
5      int a[5],i,*pa,*pb;
6      pa = a;                                  //数组名代表首地址
7      pb = &a[0];                              //数组的第一个元素的地址也代表首地址
8
9      for(i=0;i<5;i++)
10     {
11         a[i]=i;
12         printf("a[%d]=%d\n",i,a[i]);
13         printf("a[%d]=%d\n",i,*(a+i));
14         printf("a[%d]=%d\n",i,*(pa+i));
15         printf("a[%d]=%d\n",i, *pb++);
16
17     //  printf("a[%d]=%d\n",i, a++);          //a为常量，不允许执行++操作
18     }
19     printf("\n");
20 }
```

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
  - 存放指针的数组——指针数组
  - 指向数组元素的指针
  - 指向数组的指针——数组指针（行指针）
- 函数与指针
- 其它特殊指针

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

指向数组元素的指针

- 数组元素在内存中的起始地址称为数组元素的指针

```
int a[10]={1,2,3,4};
int *p=a;                    //p为数组元素的指针
```

- 数组在内存中的起始地址称为数组的指针

```
int (*q)[10] = &a;   //q为指向数组的指针
```

a为一维数组名，代表数组第一个元素的指针

&a：对a取地址，代表将整个数组当作一个整体，将这个地址付给：q——数组指针

指向数组元素的指针

➢ 07.avr_point3.c

```
1  #include <stdio.h>
2  //数组指针
3  int main()
4  {
5      int a[5]={1,2,3,4,5};
6      int (*p)[5] = &a;          //定义一个数组指针指向一个一维数组
7
8      printf("%d\n",    *( (int *)(p+1) -1 )     );
9      return 0;
10 }
```

数组指针

➢ 指向二维数组的数组指针

```
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
int (*p)[4] = a;
```

p等价于指向二维数组第0行，可完全代替a的作用.

➢ 二维数组名

二维数组名是一个二级指针

a代表了二维数组第0行的行地址

a+1代表了第一行的行地址

*(a+1)代表了第1行第0列元素的地址

*(a+1)+2代表了第1行第2列元素的地址

((a+1)+2)代表了第1行第2列元素

多维数组与指针

表 示 形 式	含义
a	数组名，指向一维数组a[0] 即0行首地址
a+1、&a[1]	1行首地址
*(a+0)、*a 、a[0]	0行0列元素地址
(a+1)、((a+1)+0)、a[1]	1行0列元素a[1][0]的地址
*(a+1)+2、a[1]+2、&a[1][2]	1行2列元素a[1][2] 的地址
*(a[1]+2) *(*(a+1)+2) a[1][2]	1行2列元素a[1][2]的值

数组指针

08.avr_point4.c

```
#include <stdio.h">
//数组指针
int a[3][5] = {
                {1, 3, 5, 7, 20},
                {9,11,13,15, 21},
                {17,19,21,23,22}
              };
int main()
{
    char i;
    int (*p)[5] = a;                    //定义一个数组指针指向一个二维数组

    for( i = 0; i<5 ; i++)              //采用数组指针打印二维数组的第一行
        printf("%d\n",*(p[0]+i) );

    for( i = 0; i<5 ; i++)              //采用数组指针打印二维数组的第二行
        printf("%d\n",*(p[1]+i));
    return 0;
}
```

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
- 函数与指针
  - 指针作函数参数
  - 指针做函数返回值
  - 指向函数的指针
- 其它特殊指针

函数与指针

- 指针变量做函数的参数

09.point_fun.c

```
int main(void)
{
    int a,b;
    scanf("%d %d",&a,&b);
    if(a>b)
    {
    //  exchang1(a,b);
        exchang2(&a,&b);
    }
    printf("a=%d,b=%d\n",a,b);
    return 0;
}
```

函数与指针

```
1  #include <stdio.h>
2  //指针作函数的参数
3  void exchang1(int p1,int p2)
4  {
5      int p;
6      p = p1;
7      p1 = p2;
8      p2 = p;
9  }
10 void exchang2(int *p1,int*p2)
11 {
12     int p;
13     p = *p1;
14     *p1 = *p2;
15     *p2 = p;
16 }
```

函数与指针

- **数组名做函数的参数**

① 一维数组名作函数的参数

```
int main(void)
{
   int a[5]={3,4,7,2,3};
   printf( "%d\n" ,max(a,5));
}
```

- 以上两种写法均可以
- 数组做形参时，无需指定其下标

```
int max(int a[], int num)
int max(int *a, int num)
{
   int i,max;
   max=a[0];
   for(i=1;i<num;i++)
   {  if(a[i]>max)
         max=a[i];  }
    return max;
}
```

函数与指针

② 二维数组名作函数的参数

```
int main(void)
{
    int a[2][3]={
                    {3,4,7},
                    {2,3,7}
                };
printf( "%d\n" ,max(a,2,3));
    return 0;
}
```

```
int max(int a[][3],int a,int b )
{
    二维数组名作参数，行可以不写
    出来，列必须写出来
}
int max(int (*a)[3],int a,int b)
{
    数组指针做形参，完全等价与二
    维数组的应用
}
```

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
- **函数与指针**
  - 指针作函数参数
  - **指针做函数返回值**
  - 指向函数的指针
- 其它特殊指针

函数与指针

➢返回指针值的函数

•一个函数可以带回一个整型值、字符值、实型值等，也可以带回指针型的数据，即地址.

•这种带回指针值的函数，一般定义形式为

类型名 *函数名（参数表列）;

例如: `int *a(int x, int y);`

函数与指针

- 例: pc = (char *)malloc(100*sizeof(char));
  - 表示分配100个字节的内存空间，并强制转换为字符数组类型，函数的返回值为指向该字符数组的指针，把该指针赋予指针变量pc

```
char *pc=NULL;
pc = (char *)malloc(100*sizeof(char));
```

- 注意:
  - 在调用时要先定义一个适当的指针来接收函数的返回值，这个适当的指针其类型应为函数返回指针所指向的类型.

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
- **函数与指针**
  - 指针作函数参数
  - 指针做函数返回值
  - **指向函数的指针**
- 其它特殊指针

函数与指针

➢ 指向函数的指针（函数指针）

一个函数在编译时被分配一个入口地址，这个地址就称为**函数的指针**，函数名代表函数的入口地址.

这一点和数组一样，因此我们可以用一个指针变量来存放这个入口地址，然后通过该指针变量调用函数.

```
int max (int x, int y)
int c;
c = max(a,b);
```

- 这是通常用的方法，我们也可以定义一个函数指针，通过指针来调用这个函数.

函数与指针

- 例如:

```
int (*p)(int,int);//指向函数指针变量的定义形式
p = max;      //将函数的入口地址赋给函数指针变量p
c = (*p)(a,b);    //调用max函数
```

- 说明:

① p不是固定指向哪个函数的，而是专门用来存放函数入口地址的变量,在程序中把那个函数的入口地址赋给它，它就指向那个函数.

- p不能像指向变量的指针变量一样进行p++,p--等无意义的操作.

函数与指针

- 函数指针的应用——回调函数

  函数指针变量常见的用途之一是把指针作为参数传递到其他函数.

  指向函数的指针也可以作为参数，以实现函数地址的传递，这样就能够在被调用的函数中使用实参函数.

- 例:

  设一个函数process，在调用它的时候，根据传入参数的不同实现不同的功能.

  输入 a 和 b 两个数，第一次调用process时找出 a 和 b 中大者，第二次找出其中小者，第三次求 a 与 b 之和

函数与指针

10. point_fun

```
int main ()
{
    int a,b,result;

    printf ("enter a and b:");
    scanf("%d %d",&a,&b);

    printf("max=");
    result=process(a,b,max);
    printf("%d\n",result);

    printf("min=");
    result=process(a,b,min);
    printf("%d\n",result);

    printf("sum=");
    result=process(a,b,add);
    printf("%d\n",result);
    return 0;
}
```

函数与指针

```
int max( int x, int y)
{
    return x>y?x:y;
}

int min( int x ,int y)
{
    return x<y?x:y;
}

int add(int x, int y )
{
    return x+y;
}
```

函数与指针

```
int  process(int x,int y,int (*fun)(int,int))
{
    int result;
    result = (*fun)(x,y);
    return result;
}
```

- 例：

process称为回调函数，其实process并没有直接调用max min add函数，而是通过fun函数指针接收相应函数首地址，进行调用，调用后把结果返回给主调函数main

大纲

- 有关内存的那点事
- 指针的相关概念—指针与指针变量
- 变量与指针
- 字符串与指针
- 数组与指针
- 函数与指针
- 其它特殊指针
  - main函数带参
  - 指向指针的指针
  - void *指针

函数与指针

- main函数带参
- main函数可以接收来自操作系统或者其它应用传递的参数

```
int main(int argc ,char *argv[])
```

argc代表参数的个数

argv存放各参数首地址（系统把每个参数当作一个字符串放在系统缓冲区，把首地址放在指针数组中）

函数与指针

➢ 11.main_param.c

```
1  #include <stdio.h>
2  int main (int argc ,char *argv[])
3  {
4      int i;
5      printf("argc=%d\n",argc);
6      for( i = 0 ; i < argc; i++ )
7      {
8          printf("argv[%d]=%s\n",i,argv[i]);
9      }
10     return 0;
11 }
```

函数与指针

- **char **p 指向指针的指针**

一个变量有自己地址，我们可以用一个指针变量指向他，指针变量同样也是变量，我们也可以定义另一个指针指向这个指针变量，称之为指向指针的指针

```
int a;
int *p = &a;
int *q = &p;
```

- 常用于通过函数改变指针的指向，见后例

函数与指针

12.point_point.c

在函数内部使用malloc申请空间

使用参数带回申请到空间的首地址

在子函数内部改变主调函数定义指针的指向

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  void GetMemory1(char *p,int num)
5  {
6      p = (char *)malloc(100);
7  }
8  void GetMemory2(char **p, int num)
9  {
10     *p = (char *)malloc(num);
11 }
12 int main (void)
13 {
14     char *str = NULL;
15     GetMemory1(str,100);
16     //GetMemory2(&str, 100);
17     strcpy(str, "hello world");
18     printf("%s\n", str);
19     free(str);
20     return 0;
21 }
```

函数与指针

- **void类型的指针 void***

  void指针是一种很特别的指针，并不指定它是指向哪一种类型的数据，而是根据需要转换为所需数据类型.

```
int a=0;
float b=0;
void *p=&a;
void *p=&b;
```

- **常用于函数的返回值和参数**

  用于不确定类型指针指向

  - void *malloc(unsigned int num_bytes);
  - void *memset(void *mem, char ch, size_t n);

凌阳教育
www.sunplusedu.com

值得信赖的教育品牌

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com