指针数组和数组指针

##

数组类型

> 数组大小+元素类型 共同决定
>
> int array[5]的类型为int[5]

```
typedef type(name)[size];   //定义格式

typedef int(AINT5)[5];
typedef float(AFLOAT10)[10];


//数组定义
AINT5 iArray;
AFLOAT10 fArray;
```

数组指针

> 用于指向一个数组

```
//定义方式2种
ArrayType* pointer;
type (*pointer)[n];

//类比
int i;
int *pi = &i;

ArrayType *p;
int array[5];
p= &array;  //数组指针
```

```
#include <stdio.h>

typedef int(AINT5)[5];
typedef float(AFLOAT10)[10];
typedef char(ACHAR9)[9];

int main()
{
 AINT5 a1;
 float fArray[10];
 AFLOAT10* pf = &fArray;
 ACHAR9 cArray;
 char(*pc)[9] = &cArray;
 char(*pcw)[4] = cArray;
```

```
 int i = 0;

 printf("%d, %d\n", sizeof(AINT5), sizeof(a1));

 for(i=0; i<10; i++)
 {
     (*pf)[i] = i;
 }

 for(i=0; i<10; i++)
 {
     printf("%f\n", fArray[i]);
 }

 printf("%0X, %0X, %0X\n", &cArray, pc+1, pcw+1);
}

//pc加的是整个数组的长度
//pcw加的是一个元素的长度
```

指针数组

```
type* pArray[n];
//数组中每个元素为一个指针
```

```
#include <stdio.h>
#include <string.h>

//这里table就是指针数组，size是因为编译器隐藏了数组长度所以需要
int lookup_keyword(const char* key, const char* table[], const int size)
{
 int ret = -1;

 int i = 0;

 for(i=0; i<size; i++)
 {
     if( strcmp(key, table[i]) == 0 )
     {
         ret = i;
         break;
     }
 }

 return ret;
}

#define DIM(a) (sizeof(a)/sizeof(*a))

int main()
{
```

```
 const char* keyword[] = {
         "do",
         "for",
         "if",
         "register",
         "return",
         "switch",
         "while",
         "case",
         "static"
 };

 printf("%d\n", lookup_keyword("return", keyword, DIM(keyword)));
 printf("%d\n", lookup_keyword("main", keyword, DIM(keyword)));
}
```

main函数的参数

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

//命令行参数个数，参数数组，环境变量数组
//后两个是指针数组
```

```
#include <stdio.h>

int main(int argc, char* argv[], char* env[])
{
 int i = 0;
 printf("============== Begin argv ==============\n");
 for (i = 0; i < argc; i++)
 {
     printf("%s\n", argv[i]);
 }
 printf("============== End argv ==============\n");
 printf("\n");
 printf("\n");
 printf("\n");
 printf("============== Begin env ==============\n");

 for (i = 0; env[i] != NULL; i++)
 {
     printf("%s\n", env[i]); //打印出系统的所有环境变量
 }
 printf("============== End env ==============\n");
}
```

指向指针的指针

> 因为指针本质上也是变量
>
> 对于指针也存在传值和传址调用

二维数组

> 二维数组名为首元素的地址（即数组的地址），为数组指针类型
>
> 在内存中以一维方式排布。
>
> c语言本质上只有一维数组

```
int m[2][5];//m的类型为(int*)[5]


//等价格式
a[i][j];
*(*(a+i)+j);
```

数组参数

> c语言在函数传递时如果拷贝整个数组，执行效率会大大下降
>
> 所以在向函数传递数组时，将数组名看作常量指针传数组首元素地址。

```
//类比

//二维数组可以看作时一维数组
//二维数组中的每个元素是一维数组
//二维数组的第一维参数可以省略


void f(int a[5]) ;   void f(int a[]);   void f(int* a);  //整行等价
void g(int a[3][3]);  void g(int a[][3]);   void g(int (*a)[3]); //等价
```

数组参数	等价的指针参数
一维数组 float a[5]	指针 float * a
指针数组 int * a[5]	指针的指针 int ** a
二维数组 char a [3] [4]	数组的指针 char (*a)[4]

> 注意：
>
> c语言无法向一个函数传递任意的多维数组，类型大小都要匹配
>
> 必须提供除第一维之外的所有维长度
>
> 限制：

一维数组参数：必须提供一个标示数组结束位置的长度信息

二维数组参数：不能直接传递给函数

三维或更多维：无法使用

```
void access(int(*a)[3], int row)
{
    int col = sizeof(*a) / sizeof(int);
    int i = 0;
    int j = 0;

    printf("sizeof(a)=%d\n", sizeof(a));

    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col ;j++)
        {
            printf("%d\n", a[i][j]);
        }
    }
}

int main()
{
    int a[3][3] = { {0,1,2},{3,4,5},{6,7,8} };
    access(a, 3);
}
```

函数与指针

函数类型： 返回值+参数类型+参数个数 共同决定

```
int add(int i,int j);
// int (int ,int)类型
```

通过typedef为函数类型**重命名**

```
typedef type name(parameter list);

//举例
typedef int f(int,int);
typedef void p(int);
```

函数指针

函数指针用于指向一个函数

函数名是执行函数体的入口地址，所以有无&是一样的

定义方式2种

```
FuncType* pointer;
type(* pointer)(parameter list);
```

```
typedef int (FUNC)(int); //函数类型定义

int test(int i)
{
    return i * i;
}

void f()
{
    printf("call f()\n");
}

int main()
{
    //写法等价
    FUNC* pt = test;  //pt为指向test的函数指针
    void (*pf)() = &f; //pf为指向f的函数指针

    //等价写法，因为函数名本身就是地址
    pf();
    (*pf)();

    printf("function pointer call: %d\n", pt(2));
}
```

回调函数

利用函数指针实现的一种调用机制

回调机制：

1. 调用者不知道具体事件发生的时候需要调用的具体函数
2. 被调函数不知道何时被调用，只知道被调用后需要完成的任务
3. 当具体事件发生时，调用者通过函数指针调用具体函数

调用者和被调用者分开，互不依赖。有利于模块化。

```
#include <stdio.h>

typedef int(*FUNCTION)(int);

int g(int n, FUNCTION f)
{
    int i = 0;
    int ret = 0;

    for(i=1; i<=n; i++)
    {
        ret += i*f(i);
    }
```

```
    return ret;
}

int f1(int x)
{
    return x + 1;
}

int f2(int x)
{
    return 2*x - 1;
}

int f3(int x)
{
    return -x;
}

int main()
{
    printf("x * f1(x): %d\n", g(3, f1));
    printf("x * f2(x): %d\n", g(3, f2));
    printf("x * f3(x): %d\n", g(3, f3));
}
```

指针阅读技巧

右左法则

1.从最里层的圆括号中未定义的标示符看起

2.首先往右看，再往左看

3.当遇到圆括号或者方括号时可以确定部分类型，并调转方向

4.重复2,3步骤，直到阅读结束

```
int (*func)(int*); //函数指针，指向类型为 int (int*)


int (*p2)(int*, int (*f)(int*)); //p2函数指针，f函数指针

int (*p3[5])(int*);  //p3指针数组，每个元素都是函数指针

int (*(*p4)[5])(int*);//p4是数组指针，数组的每个元素都是函数指针

int (*(*p5)(int*))[5];//p5是
```