

一、函数指针

1. 简单介绍
2. 回忆函数
3. 函数地址
4. 函数指针
5. 案例
 - (1) 案例一
 - (2) 案例二
 - (3) 案例三
 - (4) 案例四代码1
代码2
误区
6. 补充

二、函数指针数组

1. 定义
2. 补充
3. 案例
 - (1) 案例一
 - (2) 案例二
4. 转移表（计算器实例）
 - (1) 一般写法
 - (2) 改进

三、指向函数指针数组的指针

定义

一、函数指针

1. 简单介绍

一个函数如何设计，才能合理地接收参数？

以前学习过，**数组指针** 就是 **指向数组的指针**。

那么，**函数指针** 就是 **指向函数的指针**。

2. 回忆函数

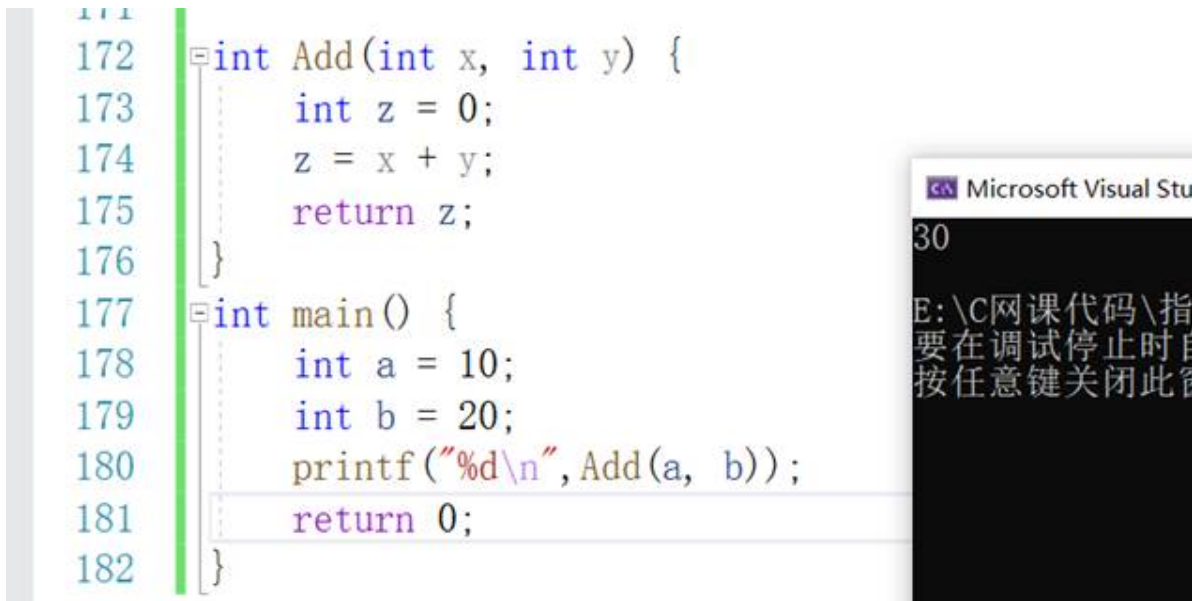
回忆一下我们之前说的函数，写一个加法函数：

```

int Add(int x,int y){
    int z=0;
    z=x+y;
    return z;
}
int main(){
    int a=10;
    int b=20;
    printf("%d\n",Add(a,b));
    return 0;
}

```

输出结果如下：



3.函数地址

既然函数指针是用来存放函数地址的，函数也有地址吗？

我们来打印一下：

```
printf("%p\n",&Add);
```

具体代码如下：

```

int Add(int x,int y){
    int z=0;
    z=x+y;
    return z;
}
int main(){
    int a=10;
    int b=20;
    printf("%p\n",&Add);
    return 0;
}

```

输出看一下，果然输出了一个地址。

这个地址，就是函数Add的地址。

```
171
172 int Add(int x, int y) {
173     int z = 0;
174     z = x + y;
175     return z;
176 }
177 int main() {
178     int a = 10;
179     int b = 20;
180     //printf("%d\n", Add(a, b));
181     printf("%p\n", &Add);
182     return 0;
183 }
```

Microsoft Visual Studio 调试控制台

007310B4

E:\C网课代码\指针进阶 (3)
要在调试停止时自动关闭控制台
按任意键关闭此窗口...

我们之前说数组的时候：

```
int arr[10] = { 0 };
```

&arr 与 arr 拿到的结果是一样的。

那么在函数中是否也存在一样的道理？

我们来打印一下：

```
printf("%p\n", &Add);
printf("%p\n", Add);
```

看一下输出结果：

```
172 int Add(int x, int y) {
173     int z = 0;
174     z = x + y;
175     return z;
176 }
177 int main() {
178     int a = 10;
179     int b = 20;
180     //printf("%d\n", Add(a, b));
181     printf("%p\n", &Add);
182     printf("%p\n", Add);
183 }
```

Microsoft Visual Studio 调试控制台

00FC10B4
00FC10B4

E:\C网课代码\指针进阶
要在调试停止时自动关闭控制台
按任意键关闭此窗口...

可以看到结果是一样的。

那 Add 是函数首元素地址吗？

胡扯！哪儿有函数首元素啊！函数只有一个，数组才有首元素的概念。

在函数里面，Add 与 &Add 是一样的。

函数名 与 &函数名 都是函数的地址。

4.函数指针

当我们真的能够拿到函数地址的时候，函数地址存哪儿呢？

还是拿上面的代码举例。

要将函数名Add存起来，就需要一个指针变量pa。如下：

```
pa=Add;
```

这个指针变量pa的类型是什么呢？

在之前，我们要存数组的地址。假设要存入指针p，p的类型怎么写？

```
int arr[10] = { 0 };
```

P是一个指针，要指向数组，(*p) 表示p为指针，[] 表示指向数组，几个元素呢？10个，即：[10]。

数组每个元素的类型是什么呢？我们指向的数组每个元素是int类型。所以这样写：int (*p)[10]=&arr;

👉 那么对于函数指针的写法：

①第一种写法

这样写行吗？

```
int *pa(int,int)=Add; //不行，这里的pa就相当于一个函数名，有两个参数int，返回类型是int*
```

pa首先和后面的()结合，说明pa是个函数的函数名。

后面的两个int是参数类型，前面的int*是返回类型。这是不对的!!!

我们希望pa是个指针，指针才能存放地址。

②第二种写法

这种才是正确写法：

```
int(*pa)(int,int)=Add;
```

pa首先和*结合，才能保证它是一个指针。

后面的圆括号表示指向的是函数，函数的参数是两个整型。

后边圆括号里面，写int x与int y和int与int一样。x和y可以写可以不写，只需要把函数的参数类型（这里是int）交代清楚即可。

指向的函数的返回类型是什么？是int类型。所以最前面写上int即可。

5.案例

(1) 案例一

上面我们已经会书写函数指针了：

```
int(*pa)(int,int)=Add;
```

这里的pa究竟是不是函数指针呢？里面是否存放的是函数的地址？

pa里面如果存的是函数的地址，那*pa就可以找到那个函数。

找到那个函数后，我们就可以调用那个函数。

调用函数要传参，这里我们把2和3传进去，如下：

```
(*pa)(2, 3)
```

再打印一下即可。

整体代码如下：


```
int Add(int x,int y){
    int z=0;
    z=x+y;
    return z;
}
int main(){
    int a=10;
    int b=20;
    int(*pa)(int,int)=Add; //将Add函数名赋值给pa指针
    printf("%d\n",(*pa)(2, 3)); //pa找到函数
    return 0;
}
```

输出结果：

```

172 int Add(int x, int y) {
173     int z = 0;
174     z = x + y;
175     return z;
176 }
177 int main() {
178     int a = 10;
179     int b = 20;
180
181     int(*pa)(int, int) = Add; //pa里面如果存的是函数的地址，那*pa就可以找到那个函数，
182     //我们来调用那个函数，调用函数要传参，把2和3传进去
183     printf("%d\n", (*pa)(2, 3));
184
185

```



(2) 案例二

函数指针是指向函数的指针，是存放函数地址的一个指针。

不同函数的地址存起来，对应函数指针的定义方式应该也不相同。

我们来举个例子：

```

void Print(char* str) {
    printf("%s\n", str);    //把str指向的字符串打印出来
}
int main() {
    void(*p)(char*) = Print;    //Print函数名就是地址，存进指针变量p里面
    (*p)("hello");    //p里面存的是函数的地址，解引用找到该函数
    //调用函数，将字符串传给字符指针
    return 0;
}

```

`void(*p)(char*) = Print;`这行代码的意思：

`(*p)` 表示p是指针，后面圆括号表示指向的是函数，`char*` 表示指向函数的参数类型，返回类型是 `void`。

输出结果：

```

197 void Print(char* str) {
198     printf("%s\n", str); //把str指向的字符串打印出来
199 }
200 int main() {
201     void(*p)(char*) = Print;
202     //(*p)表示指针，后面圆括号表示指向的是函数，char*表示指向函数的参数类型，返回类型是void
203     (*p)("hello"); //p里面存的是函数的地址，解引用找到该函数
204
205     return 0;
206 }

```



(3) 案例三

再来看一段代码，巩固一下。

如下：

```
void test() {
    printf("hehe\n");
}
//下面pfun1和pfun2哪个有能力存放test函数的地址?
void (*pfun1)();
void* pfun2();
```

能存储函数的地址，就要求pfun1或者pfun2是指针，那哪个是指针？

①pfun1先与 * 结合，说明它是指针，指针可以存放地址，没有问题。指向函数的参数是无参，返回类型是void。

②pfun2首先和 () 结合，说明它是个函数，函数参数是无，返回类型是void。pfun2不是指针，是个函数名而已。

所以pfun1有能力存放test函数的地址。

(4) 案例四

我们再来看两段有趣的代码：

代码1

```
(* (void(*)())0) ();
```

🚗 分析：

先来看一下它的括号如何配对。

经过分析，应该是以下的配对：

(*(void(*)())0) ();

<1> 红色括号中间 void(*)()，* 代表指针类型。指针指向后边的蓝色括号部分，说明它是函数，返回类型是void。

那么这个 void(*)() 部分，就是**函数指针类型**。

将这个类型放进红色括号里面，叫**强制类型转换**。（一个括号里面放一个类型，叫强制类型转换）

<2> 这个强制类型转换，放在了0前面，说明是把0进行强制类型转换。

原来的0是整数类型，现在将它强制类型转换成函数指针类型。意味着想要把**0当成某函数的地址**。

<3> 将0强制转换成函数的地址之后，前面有一个解引用（就是最前面的 * 号）。

解引用之后，就找到了这个函数。

这个函数是无参的，返回类型是void---> void(*)()。

<4> 接下来去调用这个函数（最后的橙色括号就是调用函数），没有传参。因为指向的函数是无参的。

🌿 总结

总而言之，这行代码就是在调用函数，调用0地址处，参数为无参，返回类型是void的函数。

把0强制类型转换为 `void(*)()` 函数指针类型，0就是一个函数的地址。然后解引用，调用0地址处的该函数。

代码2

```
void(*signal(int,void(*) (int)))(int);
```

🚗 分析：

先来看一下它的括号如何配对。

经过分析，应该是以下的配对：

void(*signal(int,void(*) (int)))(int);

<1> 首先看到了 `signal`，是个名字，后边一对圆括号，说明是一个函数。

函数有两个参数，第一个是 `int`，**整数类型**；

第二个是 `void(*) (int)`，`*` 表示它是指针，指向函数的参数类型是 `int`，返回类型是 `void`，是一个**函数指针类型**。

<2> 经过上面的分析，这个 `signal` 函数有两个参数，第一个是整型，第二个是函数指针类型。

那么，函数名确定了，参数也确定了。

<3> 函数的返回类型是什么呢？

之前接触过函数：`int Add(int,int)`：函数名叫Add，参数有两个，都是int类型。去掉函数名和参数，剩下的就是函数的返回类型。所以Add函数的返回类型是int。

之前我们看到这样的代码也是类似：

```
(1) int(*p)[10]=&arr;
```

//P是指针变量，指向的是一个数组，数组元素是int。我们把p去掉的话，剩下的就是它的类型。P是指向数组的指针类型。

```
(2) void (*p)(char*)=Print;
```

//把p变量去掉，剩下的就是类型。是指针，指向的是函数。

再回到signal函数，将函数名 (`signal`) 和参数 (`int,void(*) (int)`) 去掉，剩下的 `void(*) (int)` 就是函数返回类型。

<4> 对于 `void(*) (int)` , `*` 表示它是指针 (`*` 号在括号里面, 所以先于 `*` 结合), 指向的是一个函数 (因为 (`*`) 后边有括号), 参数是 `int` 类型 (因为后边括号里面是 `int`), 返回类型是 `void`。

所以函数返回类型是**函数指针类型**。

📌 总结

总而言之, 这行代码就是在**函数声明**。(告诉我们, 函数名、参数、返回类型分别是什么)

`signal` 是函数名, 有两个参数 (整型和函数指针类型), 函数的返回类型是函数指针类型。

误区

🧑 注意

对于第二个代码, 有的同学可能这样书写。

比如现在让大家写一个 `signal` 函数, 有两个参数, 分别是 `int` 类型和 `void(*) (int)` 类型, 函数返回类型是 `void(*) (int)` 类型。

可能有的同学会这样写 `signal` 函数:

```
void(*) (int) signal(int, void(*) (int))
```

这种写法是不对的!

函数指针的返回类型不能就这样写在前面。

必须要按照规范, `*` 号要靠近函数名, 后半部分 (即 `(int)`) 要放在最后面。如下:

```
void(*signal(int, void(*) (int))) (int);
```

这种正确写法不容易理解。

🐱 精简

那能不能精简一些呢?

以前学过一个关键字 `typedef`, 这个关键字可以让某些类型简单一些。如下:

```
typedef unsigned int uint; // unsigned int是类型, uint是新取的名字
```

我们现在觉得 `signal` 函数的参数类型 (`void(*) (int)`) 有一点复杂, 可以给它取一个新名字 `pfun_t`。

注意, 这样写是不对的: (上面已经说明了这个错误)

```
typedef void(*) (int) pfun_t;
```

应该这样写: (将名字靠近 `*` 号)

```
typedef void(*pfun_t) (int);
```

现在的 `pfun_t` 就是函数指针类型了。

`signal` 返回的是函数指针类型，现在重新命名为了 `pfun_t`，那么 `signal` 函数就可以这样精简。

如下：

```
pfun_t signal(int, pfun_t);
```

综上，这一行代码可以精简为两行代码：

```
//第一种写法
void(*signal(int,void(*)(int)))(int);

//第二种写法
typedef void(*pfun_t)(int);
pfun_t signal(int, pfun_t);
```

6.补充

在上面的【案例一】，我们看过这样的代码：

```
int Add(int x,int y){
    int z=0;
    z=x+y;
    return z;
}
int main(){
    int a=10;
    int b=20;
    int(*pa)(int,int)=Add; //将Add函数名赋值给pa指针
    printf("%d\n",(*pa)(2, 3)); // *pa找到函数
    return 0;
}
```

`pa` 里面存的是 `Add` 函数的地址，调用的时候是 `*(pa)`，解引用找到这个函数。

❓ 这里的 `*` 号有什么意义呢？

比如我们这里给它加一颗 `*`，或者加两颗 `*`。

如下：

```
printf("%d\n",(**pa)(2, 3));
printf("%d\n",(**pa)(2, 3));
```

这个输出结果是啥呢？

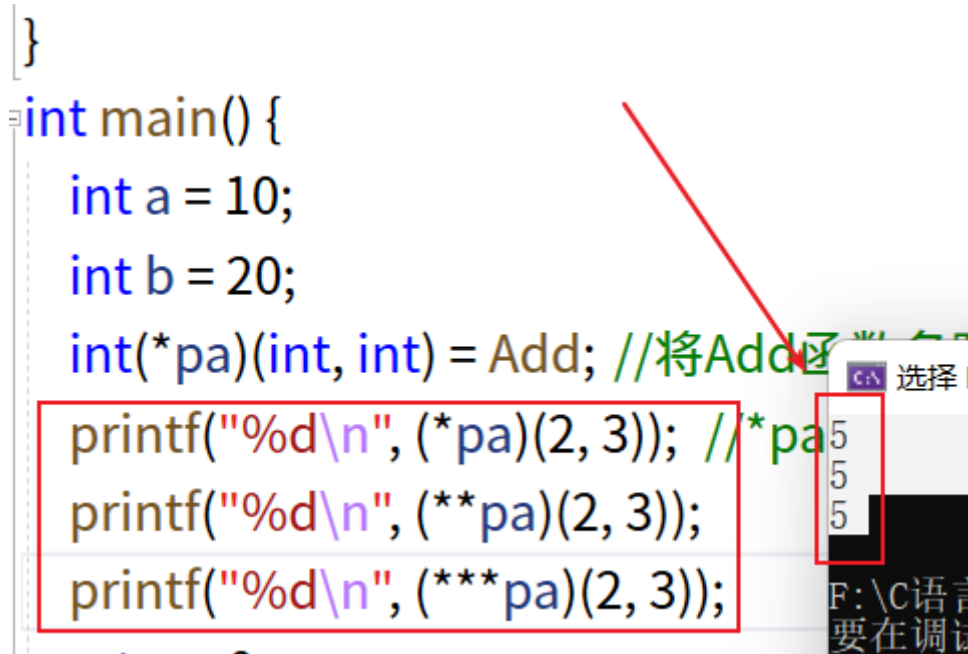
看一下：

```

}

int main() {
    int a = 10;
    int b = 20;
    int(*pa)(int, int) = Add; //将Add函数名赋值给pa
    printf("%d\n", (*pa)(2, 3)); //*pa找到函数
    printf("%d\n", (**pa)(2, 3));
    printf("%d\n", (***)pa)(2, 3));
}

```



可以看见，三个结果都是5，这说明了什么？

说明这个 * 只是摆设！

那我们现在将 * 号去掉：

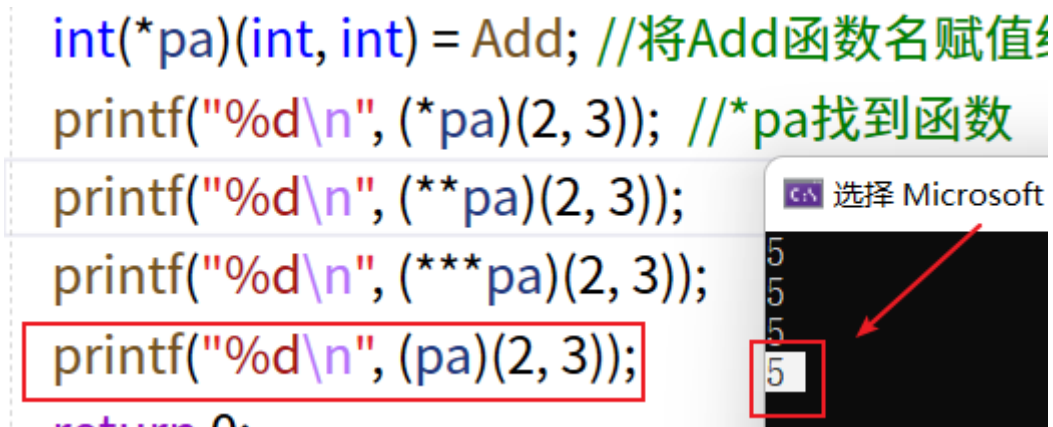
```
printf("%d\n", (pa)(2, 3));
```

再次输出看一下：

```

int(*pa)(int, int) = Add; //将Add函数名赋值给pa
printf("%d\n", (*pa)(2, 3)); //*pa找到函数
printf("%d\n", (**pa)(2, 3));
printf("%d\n", (***)pa)(2, 3));
printf("%d\n", (pa)(2, 3));
return 0;

```



📦 总结

如果 `pa` 是函数指针，那么调用的时候，可以解引用调用函数，也可以不解引用。

所以，一般是以下两种写法：

① `pa` 是指针，想要找到它所指向的函数并且调用，先解引用，然后再传参调用。

这是我们对指针的最初印象，就是：指针解引用才能找到它指向的对象。

即：

```
int(*pa)(int,int)=Add; //将Add函数名赋值给pa指针
printf("%d\n",(*pa)(2,3)); //第一种（不可以将*pa的括号去掉）
//如果*pa去掉括号，pa就会先与后面的(2,3)结合，算出5，5再解引用，这时候就会出问题-->非法的间接寻址）
```

②将Add函数传给了pa，说明Add和pa是一回事。

没有pa的时候，调用函数是这样 `Add(2,3)`，函数名是地址，pa也是存放函数地址的。

所以也可以这样写：

```
int(*pa)(int,int)=Add; //将Add函数名赋值给pa指针
printf("%d\n",Add(2,3));
printf("%d\n",(*pa)(2,3)); //第二种（可以将pa的括号去掉）
```

对于函数来说，`*`没有什么价值，但是写上之后也是有意义的。写的时候，必须将`*pa`括号括起来。

写上之后，比较好理解。说明是解引用找到了这个对应的函数，然后去调用它。

二、函数指针数组

1.定义

数组是一个存放相同类型数据的存储空间，我们已经学了指针数组，详情请见[指针进阶之数组指针和指针数组](#)

比如：

```
int* arr[10]; //arr是一个指针数组，数组里面10个元素，每个元素是int*类型
```

现在要把**函数的地址存到一个数组中**，这个数组就叫**函数指针数组**。

那函数指针数组如何定义呢？

举个例子。

现在有一个函数Add：

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

将函数Add的地址存起来：

```
int (*pa)(int,int)=Add; //pa是一个函数指针，指向的函数参数类型有两个，都是int类型，返回类型也是int
```

pa存放的就是Add地址。

现在，不仅仅有一个加法函数Add，还有一个减法函数Sub。

```
int Sub(int x,int y){  
    return x-y;  
}
```

还有一个乘法函数Mul。

```
int Mul(int x,int y){  
    return x*y;  
}
```

还有一个除法函数Div。

```
int Div(int x,int y){  
    return x/y;  
}
```

这几个函数地址类型一模一样。

所以pa指针也可以存放Sub, Mul, Div。

但是现在pa里面只能存放一个地址，我们想要把四个地址都存起来。

这时候就需要一个数组，这个数组可以存放四个函数的地址。即：**函数指针数组**。

那么函数指针数组咋写呢？假设数组名为parr。

很简单，将之前的代码改一下，让parr与[]结合，成为数组即可。

如下：

```
int (*parr[4])(int,int)
```

parr先与[]结合，是一个数组，数组里面有4个元素。

将数组名parr和元素个数[4]去掉，剩下的就是数组元素类型，即：函数指针类型int(*)
(int,int)。

此时的parr就是一个存放4个函数指针的数组。

可以初始化一下：

```
int (*parr[4])(int,int)={Add,Sub,Mul,Div}; //parr为函数指针数组
```

2.补充

来说一下**函数指针数组**的正确写法，不要写错了。

①代码1

```
int (*parr1[10])();
```

这种写法是**正确**的。

`parr1` 先与 `[]` 结合，说明 `parr1` 是一个数组，数组内容是 `int(*)()` 类型的函数指针，该函数无参，返回类型是 `int`。

②代码2

```
int *parr2[10]();
```

这个代码啥也不是，**语法错误**。

不要写这样的代码。

③代码3

```
int (*)() parr3[10];
```

这种写法也是**错误**的。

`parr3[10]` 应该放在 `*` 号旁边。

3.案例

(1) 案例一

函数指针数组如何使用呢？

举个例子吧。

还是拿上面的函数举例子。

现在有这么多函数，用 `parr` 函数指针数组存放起来了。

```
int Add(int x,int y){
    return x+y;
}
int Sub(int x,int y){
    return x-y;
}
int Mul(int x,int y){
    return x*y;
```

```

}
int Div(int x,int y){
    return x/y;
}
int main(){
    int (*parr[4])(int,int)={Add,Sub,Mul,Div}; //parr为函数指针数组
    return 0;
}

```

找到数组parr的每个元素： `parr[i]`。

数组parr的每个元素是函数地址，想要调用它，就需要解引用（这里就不解引用使用了，上面[误区](#)说明过）。

再给参数赋值，比如： `parr[i](2,3)`。

```

int main(){
    int (*parr[4])(int,int)={Add,Sub,Mul,Div}; //parr为函数指针数组
    int i=0;
    for(i=0;i<4;i++){
        printf("%d\n",parr[i](2,3));
    }
    return 0;
}

```

当 `i=0` 的时候，调用Add函数，算出结果5。

当 `i=1` 的时候，调用Sub函数，算出结果-1。

当 `i=2` 的时候，调用Mul函数，算出结果6。

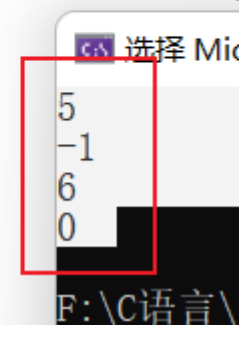
当 `i=3` 的时候，调用Div函数，算出结果0。

来看一下输出结果：

```

int main() {
    int (*parr[4])(int, int) = { Add,Sub,Mul,Div };
    int i = 0;
    for (i = 0; i < 4; i++) {
        printf("%d\n", parr[i](2, 3));
    }
}

```



(2) 案例二

现在有一个函数 `my_strcpy`，参数类型有 `char* dest` 和 `const char* src`，返回类型是 `char*`。

```
char* my_strcpy(char* dest, const char* src){  
  
}
```

💡需求

<1> 写一个函数指针 `pf`，能够指向 `my_strcpy`。

步骤：

- ①既然是指针，就在名字前面加一个 `*`，用括号括起来，即： `(*pf)`。
- ②函数指针，既然是函数，就需要在后面加一个括号，即： `(*pf)()`。
- ③括号里面写上两个参数，即： `(*pf)(char*, const char*)`。
- ④最后在最前面写上函数返回类型 `char*`，即： `char* (*pf)(char*, const char*)`。

```
char* (*pf)(char*, const char*)
```

<2> 写一个函数指针数组 `pfArr`，能够存放4个 `my_strcpy` 函数的地址。

步骤：

- ①既然是数组，就先在名字后面写上 `[]`，表示数组，数组4个元素，所以这样写： `pfArr[4]`。
- ②数组里面元素类型是函数指针类型，该函数两个参数 `char*` 和 `const char*`，返回类型是 `char*`，

所以函数指针这样写的： `char* (*)(char*, const char*)`。

- ③最后将数组名字和元素个数，放在 `*` 后面即可。即： `char* (* pfArr[4])(char*, const char*)`。

```
char* (* pfArr[4])(char*, const char*)
```

4.转移表（计算器实例）

函数指针数组在一些时候，可以让代码更加简洁。

这里举一个简单的计算器实例，让大家感受一下。

(1) 一般写法

首先设计一个 `menu` 函数:

```
void menu(){
    printf("*****\n");
    printf("** 1.add 2.sub **\n");
    printf("** 3.mul 4.div **\n");
    printf("** 0.exit **\n");
    printf("*****\n");
}
```

在主函数中, 输入一个值。

```
int main(){
    int input=0;
    do{
        menu();
        printf("请选择: >");
        scanf("%d",&input);
    }
    return 0;
}
```

再根据输入的值, 进行对应的加减乘除运算。

可以使用 `switch` 函数来进行判断, 将input的值传进去。

```
switch(input){

}
```

比如Add函数:

```
switch(input){
    case 1:
        Add();
        break;
}
```

现在来设计一下函数:

加法函数:

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

减法函数Sub。

```
int Sub(int x,int y){
    return x-y;
}
```

乘法函数Mul。

```
int Mul(int x,int y){
    return x*y;
}
```

除法函数Div。

```
int Div(int x,int y){
    return x/y;
}
```

输入操作数，并用变量x和y存储。

如下：

```
int input=0;
int x=0;
int y=0;
do{
    menu();
    printf("请选择: >");
    scanf("%d",&input);
    printf("请输入两个操作数: >");
    scanf("%d%d",&x,&y);
}
```

再将x和y放入Add函数参数中，输出打印：

```
switch(input){
    case 1:
        printf("%d\n",Add(x,y));
        break;
}
```

同样可以写出其他函数：

```
switch(input){
    case 1:
        printf("%d\n",Add(x,y));    //加法
        break;
    case 2:
        printf("%d\n",Sub(x,y));    //减法
        break;
    case 3:
        printf("%d\n",Mul(x,y));    //乘法
}
```

```

        break;
    case 4:
        printf("%d\n",Div(x,y));    //除法
        break;
    case 0:
        printf("退出\n"); //用户输入的是0，就提示退出
        break;
    default:
        printf("选择错误\n");    //输入其他值
        break;
}

```

do 后面 while 还没有写，加上：

```

int main(){
    int input=0;
    int x=0;
    int y=0;
    do{
        menu();
        printf("请选择: >");
        scanf("%d",&input);
        printf("请输入两个操作数: >");
        scanf("%d%d",&x,&y);

        switch(input){
            case 1:
                printf("%d\n",Add(x,y));    //加法
                break;
            case 2:
                printf("%d\n",Sub(x,y));    //减法
                break;
            case 3:
                printf("%d\n",Mul(x,y));    //乘法
                break;
            case 4:
                printf("%d\n",Div(x,y));    //除法
                break;
            case 0:
                printf("退出\n"); //用户输入的是0，就提示退出
                break;
            default:
                printf("选择错误\n");    //输入其他值
                break;
        }
    }while(input);
    //将input写进去，如果输入1~4，就进入do-while循环输出相应的值；如果输入0，就跳出了循环；
    //如果输入其他值，用户就重新选择
    return 0;
}

```

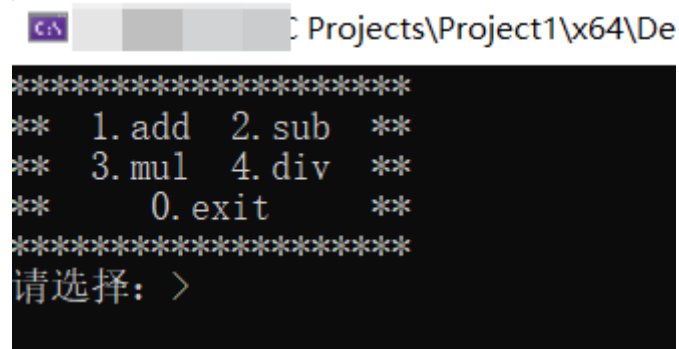
整体代码如下：

```
void menu(){
    printf("*****\n");
    printf("*** 1.add 2.sub **\n");
    printf("*** 3.mul 4.div **\n");
    printf("*** 0.exit **\n");
    printf("*****\n");
}
int Add(int x,int y){
    return x+y;
}
int Sub(int x,int y){
    return x-y;
}
int Mul(int x,int y){
    return x*y;
}
int Div(int x,int y){
    return x/y;
}
int main(){
    int input=0;
    int x=0;
    int y=0;
    do{
        menu();
        printf("请选择: >");
        scanf("%d",&input);
        printf("请输入两个操作数: >");
        scanf("%d%d",&x,&y);

        switch(input){
            case 1:
                printf("%d\n",Add(x,y));    //加法
                break;
            case 2:
                printf("%d\n",Sub(x,y));    //减法
                break;
            case 3:
                printf("%d\n",Mul(x,y));    //乘法
                break;
            case 4:
                printf("%d\n",Div(x,y));    //除法
                break;
            case 0:
                printf("退出\n"); //用户输入的是0，就提示退出
                break;
            default:
                printf("选择错误\n");    //输入其他值
                break;
        }
    }while(input);
    //将input写进去，如果输入1~4，就进入do-while循环输出相应的值；如果输入0，就跳出了循环；
    如果输入其他值，用户就重新选择
```

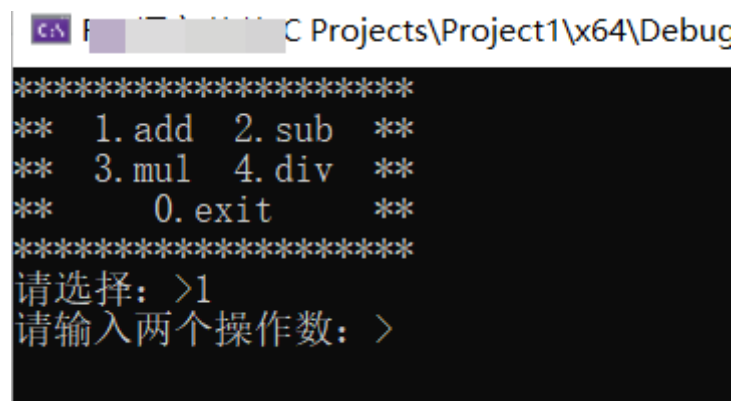
```
    return 0;  
}
```

输出看一下;



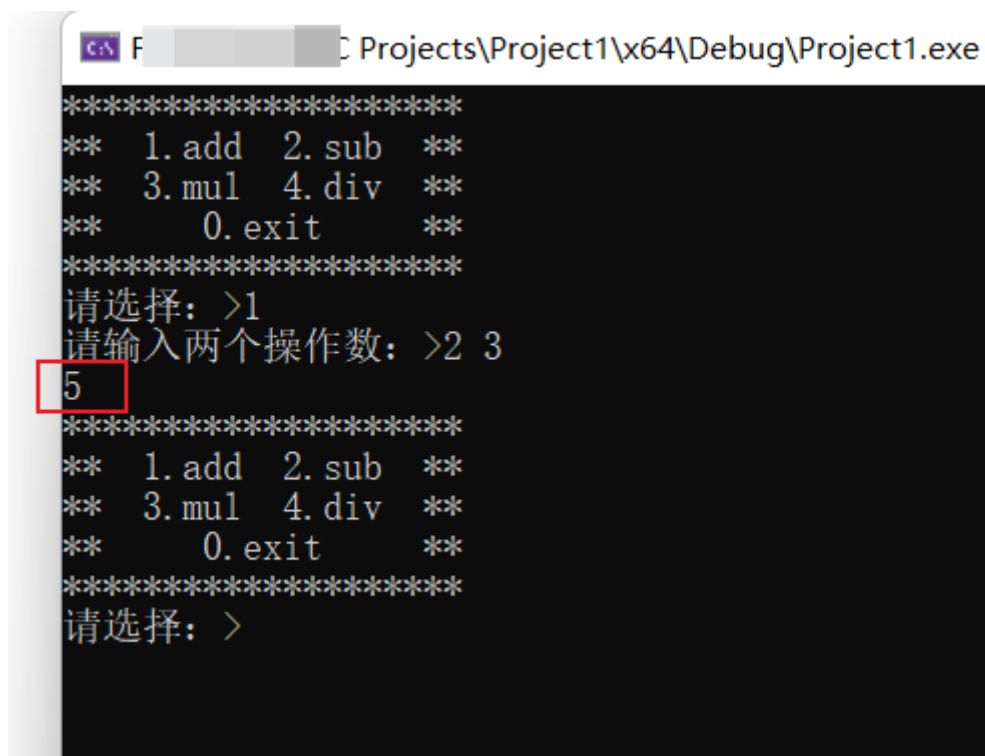
```
C:\Program Files\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\amd64\vcvarsall.bat C:\Program Files\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\amd64\cl.exe C:\Projects\Project1\x64\Debug\Project1.exe  
*****  
** 1.add 2.sub **  
** 3.mul 4.div **  
** 0.exit **  
*****  
请选择: >
```

我们输入1, 回车:



```
C:\Program Files\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\amd64\vcvarsall.bat C:\Program Files\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\amd64\cl.exe C:\Projects\Project1\x64\Debug\Project1.exe  
*****  
** 1.add 2.sub **  
** 3.mul 4.div **  
** 0.exit **  
*****  
请选择: >1  
请输入两个操作数: >
```

操作数写2 3:



```
C:\Program Files\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\amd64\vcvarsall.bat C:\Program Files\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\amd64\cl.exe C:\Projects\Project1\x64\Debug\Project1.exe  
*****  
** 1.add 2.sub **  
** 3.mul 4.div **  
** 0.exit **  
*****  
请选择: >1  
请输入两个操作数: >2 3  
5  
*****  
** 1.add 2.sub **  
** 3.mul 4.div **  
** 0.exit **  
*****  
请选择: >
```

可以看到输出了结果5。

还可以选择其他值, 比如3, 操作数是2 3:

```
C:\f\C Projects\Project1\x64\Debug\Project1.exe

*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit  **
*****
请选择: >1
请输入两个操作数: >2 3
5
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit  **
*****
请选择: >3
请输入两个操作数: >2 3
6
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit  **
*****
请选择: >
```

输入0, 退出:

```

5
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit **
*****
请选择: >3
请输入两个操作数: >2 3
6
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit **
*****
请选择: >5
请输入两个操作数: >2 3
选择错误
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit **
*****
请选择: >0
请输入两个操作数: >2 3
退出

```

其实这个代码有不好的地方，比如刚才输入了0，还要再输入操作数才能退出。

这里可以将输入代码放进case语句里面。

完整代码如下：

```

void menu(){
    printf("*****\n");
    printf("** 1.add 2.sub **\n");
    printf("** 3.mul 4.div **\n");
    printf("** 0.exit **\n");
    printf("*****\n");
}
int Add(int x,int y){
    return x+y;
}
int Sub(int x,int y){
    return x-y;
}
int Mul(int x,int y){
    return x*y;
}
int Div(int x,int y){
    return x/y;
}
int main(){

```

```

int input=0;
int x=0;
int y=0;
do{
    menu();
    printf("请选择: >");
    scanf("%d",&input);
    //printf("请输入两个操作数: >"); //移动
    //scanf("%d%d",&x,&y); //移动

    switch(input){
        case 1:
            printf("请输入两个操作数: >");
            scanf("%d%d",&x,&y);
            printf("%d\n",Add(x,y));    //加法
            break;
        case 2:
            printf("请输入两个操作数: >");
            scanf("%d%d",&x,&y);
            printf("%d\n",Sub(x,y));    //减法
            break;
        case 3:
            printf("请输入两个操作数: >");
            scanf("%d%d",&x,&y);
            printf("%d\n",Mul(x,y));    //乘法
            break;
        case 4:
            printf("请输入两个操作数: >");
            scanf("%d%d",&x,&y);
            printf("%d\n",Div(x,y));    //除法
            break;
        case 0:
            printf("退出\n"); //用户输入的是0，就提示退出
            break;
        default:
            printf("选择错误\n");    //输入其他值
            break;
    }
}while(input);
//将input写进去，如果输入1~4，就进入do-while循环输出相应的值；如果输入0，就跳出了循环；
//如果输入其他值，用户就重新选择

return 0;
}

```

(2) 改进

上面的代码，如果有特别多的函数，那么case语句就要写很长。

可以发现，case语句里面，函数调用的时候，参数都是两个。

将switch语句删掉，我们重新写一个。

```

int main(){

```



```

int input=0;
int x=0;
int y=0;
do{
    menu();
    printf("请选择: >");
    scanf("%d",&input);
    printf("请输入两个操作数: >");
    scanf("%d%d",&x,&y);

}while(input);

return 0;
}

```

现在用pfArr存储所有函数的地址。

pfArr是函数指针数组。

? 怎么书写呢?

pfArr是数组，先和 [] 结合。即：pfArr[]。

数组里面存放的是函数指针类型，即：(*)()。

该函数，有两个参数，每个参数都是int类型，返回值是int类型。所以函数指针这样写：int (*)(int,int)。

上面的案例一共有4个函数，pfArr就要存4个函数的地址。

就可以写出pfArr函数指针数组了：int (*pfArr[4])(int,int)。

这里我们初始化5个元素，因为要存一个0进去。

所以初始化这样写：

```
int (*pfArr[5])(int,int)={0,Add,Sub,Mul,Div};
```

当用户输入1, 2, 3, 4中的一个值，就可以用pfArr[input] 来访问对应元素。

比如，pfArr[1] 就可以访问Add函数。刚好和之前我们设定的，用户输入数字1，调用Add函数对应。

找到函数，然后调用函数：

```
pfArr[input](x,y)
```

用一个变量ret 接收这个值：

```
int ret=pfArr[input](x,y);
```

最后输出：

```
printf("%d\n",ret);
```

如果输入的值不是1, 2, 3, 4中的一个, 就需要退出。

所以在用户选择数字之后, 我们需要用 `if` 语句判断一下:

```
if(input>=1 && input<=4){  
  
}
```

加上之前的代码, 完整的if语句如下:

```
if(input>=1 && input<=4){  
    printf("请输入两个操作数: >");  
    scanf("%d%d",&x,&y);  
    int ret=pfArr[input](x,y);  
    printf("%d\n",ret);  
}else if(input==0){  
    printf("退出\n");  
}else{  
    printf("选择错误\n");  
}
```

 完整代码如下:

```
void menu(){  
    printf("*****\n");  
    printf("**  1.add  2.sub  **\n");  
    printf("**  3.mul  4.div  **\n");  
    printf("**    0.exit    **\n");  
    printf("*****\n");  
}  
int Add(int x,int y){  
    return x+y;  
}  
int Sub(int x,int y){  
    return x-y;  
}  
int Mul(int x,int y){  
    return x*y;  
}  
int Div(int x,int y){  
    return x/y;  
}  
int main(){  
    int input=0;  
    int x=0;  
    int y=0;  
    int (*pfArr[5])(int,int)={0,Add,Sub,Mul,Div};  
    do{  
        menu();
```

```
printf("请选择: >");
scanf("%d",&input);
if(input>=1 && input<=4){
    printf("请输入两个操作数: >");
    scanf("%d%d",&x,&y);
    int ret=pfArr[input](x,y);
    printf("%d\n",ret);
}else if(input==0){
    printf("退出\n");
}else{
    printf("选择错误\n");
}

}while(input);
//将input写进去, 如果输入1~4, 就进入do-while循环输出相应的值; 如果输入0, 就跳出了循环;
//如果输入其他值, 用户就重新选择

return 0;
}
```

输出看一下:

```
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit **
*****
请选择: >1
请输入两个操作数: >2 3
5
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit **
*****
请选择: >3
请输入两个操作数: >3 4
12
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit **
*****
请选择: >9
选择错误
*****
** 1.add 2.sub **
** 3.mul 4.div **
** 0.exit **
*****
请选择: >0
退出
```

改造之后的逻辑还是之前的逻辑。

但是，以后不管要增加什么运算。

只需要添加以下函数即可。

比如现在要增加一个异或运算。

```
int Xor(int x,int y){
    return x^y;
}
```

那么在主函数中，只需要在pfArr初始化的地方，增加Xor函数即可。如下：

```
int (*pfArr[6])(int,int)={0,Add,Sub,Mul,Div,Xor};
```

注意，菜单栏也需要增加一下这个选项：

```
void menu(){
    printf("*****\n");
    printf("** 1.add 2.sub **\n");
    printf("** 3.mul 4.div **\n");
    printf("** 5.xor 0.exit **\n");
    printf("*****\n");
}
```

if语句判断也要加一个5:

```
if(input>=1 && input<=5){
}
}
```

可以看到，用函数指针数组的形式去解决问题，会简单很多。

“转移表”的概念其实也是因为通过函数指针直接调用函数。

三、指向函数指针数组的指针

定义

指向函数指针数组的指针是一个 `指针`，指针指向一个 `数组`，数组的元素都是 `函数指针`。

来捋一下吧：

①这是整型数组 `arr`

```
int arr[10]={0};
```

②这是数组的地址 `&arr`

```
&arr
```

③将数组地址存进数组指针 `p`

```
int (*p)[10]=&arr;
```

那么：

①这是一个函数 `Add`

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

②用一个数组 `pfArr` 存放这个函数的地址：

```
int (*pfArr[4])(int,int); //pfArr是函数指针数组，里面有4个元素，每个元素是函数指针类型：int (*)(int,int)
```

③这是数组 pfArr 的地址

&pfArr

④用一个指针 ppfArr 指向数组 pfArr

如果把握不住 ppfArr 的写法，可以这样写。

首先将 pfArr 拿过来： `int (*pfArr[4])(int,int)`，然后将名字改为指针 ppfArr，即： `int (*ppfArr[4])(int,int)`。

然后 ppfArr 是一个指针，所以再加一个 * 号，括起来，即：

```
int (*( *ppfArr)[4])(int,int)=&pfArr;
```

? ppfArr 是什么呢？

首先 ppfArr 和 * 号结合，说明是一个指针。

该指针指向什么呢？

往后看，指向的是一个数组 [4]，4个元素。

每个元素的类型是什么呢？

将刚才的 *ppfArr 和 [4] 去掉，剩下的是 `int (*)(int,int)`，即：函数指针类型。

所以，ppfArr 是一个数组指针，指针指向的数组有4个元素，每个元素是函数指针类型。

再来区分一下概念：

```
int (*pf)(int,int); //函数指针
int (*pf[4])(int,int); //函数指针数组
int (*( *ppf)[4])(int,int); //函数指针数组指针
```

这个了解即可，后边可以按照需求使用。

欢迎关注，一位喜欢慢慢生活的博主。

自述文件



雨翼轻尘

110

3.3w

2.1w

原创内容

作者排名

粉丝数量



CSDN