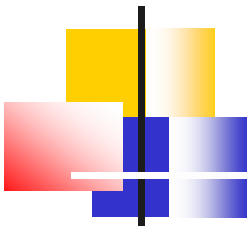


Welcome



函数接口、函数指针、泛型函数

北京亚嵌教育研究中心
©2011 AKAE

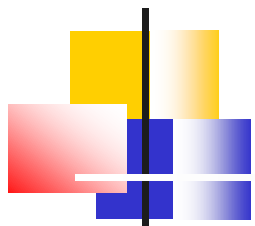


本次课程内容大纲

- 函数接口
- 函数指针的基本概念
- 深入理解函数指针
- 函数指针数组
- 回调函数
- 泛型函数

Section 1

函数接口



const关键字用法

- const是一个C语言的关键字，它限定一个变量不允许被改变（只读）。使用const在一定程度上可以提高程序的安全性和可靠性，另外，在观看别人代码的时候，清晰理解const所起的作用，对理解对方的程序也有一些帮助。

const关键字

◆ 问题：const变量 与 字符串常量

请问下面的代码有什么问题？

```
char *p = "i'm hungry!";
```

```
p[0]= 'I';
```

答案与分析：

上面的代码可能会造成内存的非法写操作，编译时没有问题，运行时出段错误。原因是字符串常量放在只读区，不可修改，若如下修改代码：

```
const char *p = "i'm hungry!";
```

编译器在编译时会提示。

const关键字

■ 总结const 与指针的各种使用方式

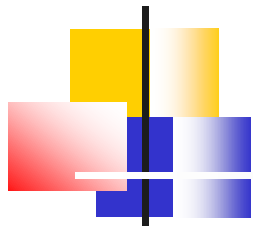
`const int nValue;` //nValue是const

`const char *pContent;` // *pContent是const, pContent可变

`char* const pContent;` // pContent是const, *pContent可变

const只是编译器对变量的约束，并不会修改变量的存储分布。

`const int n;` //n仍然存储在栈中，并不会存储到只读数据段上



const关键字

```
const char* const pContent;
```

```
//pContent和*pContent都是const
```

```
char const * pContent;
```

```
// *pContent是const, pContent可变
```

```
char* const pContent;
```

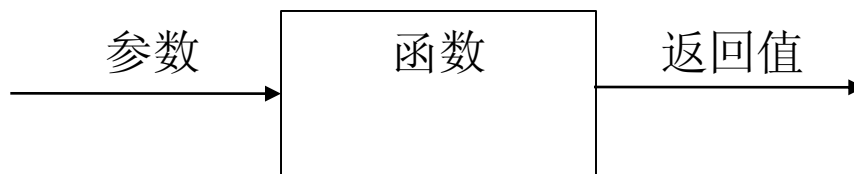
```
// pContent是const,*pContent可变
```

const关键字

- const 总结：
- 分析原则：const 左结合性。
- 作用于指针的时候：
 - const 接近类型说明符号的时候内容不可变，反之接近变量时指针不能指向其他地址。

函数接口

- 函数参数的作用：
 - 是本函数和其他函数交换数据的通道
- 函数参数的性质：
 - 通常函数参数用于接收（传入）主调函数传递来的参数



函数接口

- 函数参数的性质：

- 对于参数为指针的函数来说，由于指针可以是主调函数中某一区域的地址，那么在本函数中可以通过该指针参数修改主调函数中的数据，相当于通过函数调用**传出**了某些值。

- 当函数参数为指针时，该参数具有以下几种性质：

- 传入

- 传出

- 传入及传出

函数接口

■ 传入参数:

`void func(const char *p)`

如: `int strlen(const char *str)`

当参数指针只做传入参数使用时, 通常可以在该参数前添加`const`修饰符

调用者	实现者
1、为变量分配空间	1、只是用p地址上的数据, 不做修改
2、向变量赋值	

函数接口

■ 传出参数:

`void func(char *p)`

如: `char *strcpy(char *dest, const char *src);`

调用者	实现者
1、为变量分配空间	1、修改p地址上的数据
2、调用函数	
3、读取变量中的数据	

函数接口

■ 两层指针的参数

➤ 两层指针做传出参数

还记得哪个库函数使用了这个特性？

函数接口

■ 两层指针的参数

➤ 分配空间

```
void getmemory(char **p)
{
    *p = malloc(20);
}

int main(void)
{
    char *p = NULL;
}
```

```
int main(void)
{
    char *p = NULL;
    getmemory(&p);
    strcpy(p, "hello!");
    printf("%s\n", p);
    free(p);
    return 0;
}
```

函数接口

■ 传入/出参数:

`void func(const char *p)`

如: `void swap(int *, int *)`

调用者	实现者
1、为变量分配空间	1、读取p地址上的数据
2、向变量赋值	2、修改p地址上的内容
3、调用函数	
4、读取变量中的数据	

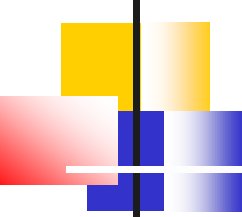
函数接口

■ 返回值是指针

返回指向已经分配内存的指针：`char *func(void)`

调用者	实现者
1、调用函数	1、规定返回值指针的类型
2、将返回值保存下来使用	2、返回指针

函数接口



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

```
static const char *msg[] = {"Sunday",
                             "Monday", "Tuesday", "Wednesday",
                             "Thursday", "Friday", "Saturday"};
```

```
char *get_a_day(int idx)
{
    static char buf[20];
    strcpy(buf, msg[idx]);
    return buf;
}
```

```
int main(void)
{
    printf( "%s %s\n" , get_a_day(0),    get_a_day(1));
    return 0;
}
```

函数接口

■ 返回值是指针

动态分配内存并返回指针: `char *alloc_unit(void)`

调用者	实现者
1、调用 <code>alloc_unit</code> 分配内存	1、规定返回值指针的类型
2、将返回值保存下来以备后用	2、 <code>alloc_unit</code> 分配内存并返回指向该内存的指针
3、调用 <code>free_unit</code> 释放内存	3、 <code>free_unit</code> 释放由 <code>alloc_unit</code> 分配的内存

切记：不可返回函数中普通的局部变量的地址

问题：函数的返回值若是指针可以返回哪些地址？

函数接口

■ 函数返回值是指针

1、可以返回传入的地址

如： `char *strcpy(char *dest, const char *src);`

2、可以返回函数中静态内存的指针

3、可以返回动态分配的内存的指针

函数接口

练习:

有如下定义: `static const char *msg[] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};`

并有如下main函数:

```
int main(void)
```

```
{
```

```
    const char *firstday = NULL, *seconday = NULL;
```

```
    //调用两次函数分别让firstday、seconday存放msg[0], msg[1]
```

```
    //自己设计get_a_day函数接口并实现调用
```

```
    get_a_day();
```

```
    printf("s\t%s\n", firstday, seconday);
```

```
    return 0;
```

```
}
```

Section 2

函数指针

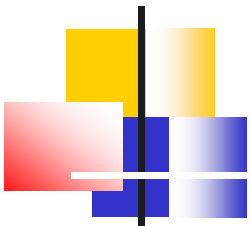
定义函数指针

■ 函数指针的一般定义形式

函数返回值类型 (*指针变量名) (函数参数列表) ;

如:

```
int (*p)(int, int);
```



定义函数指针

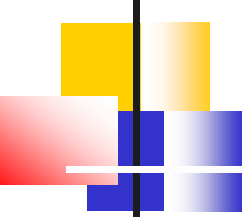
```
int (*p)(int, int);
```

定义了一个函数指针，该指针变量存放一个函数地址值，指向函数的入口地址，并且要求该函数拥有两个整形参数，一个整形返回值

有两点需要解释：

- 1、函数指针变量的名称是p
- 2、参数列表的形参名称可以不写

函数指针范例



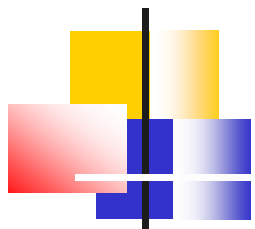
```
int func(int a, int b)
{
    return a + b;
}
int main(void)
{
    int (*p)(int, int) = func;
    printf("%d\n", (*p)(1, 2));
    return 0;
}
```

注意：

在定义函数指针的时候
要注意括号的作用
。如果例子中的定
义没有括号，写成

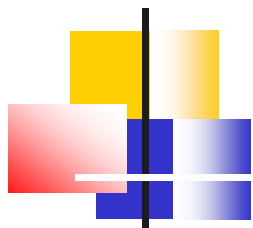
```
int *p(int, int);
```

会是什么效果呢？



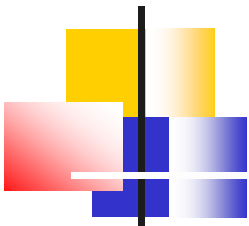
指针变量调用函数

- 定义了一个函数指针之后，我们就可以像引用变量那样引用它，可以对它进行赋值等操作，需要注意的是 任何对于函数指针的++、+、-、*、/运算都是无意义的。因为函数是一段代码，不同的函数长度不同，+、-运算结论没有合法解释。
- 定义函数指针的根本目的：**通过指针调用函数**



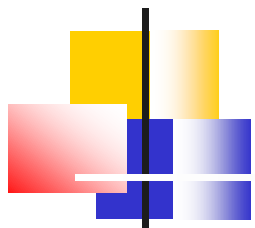
函数指针的本质

- 函数的本质是一段可以执行的代码，函数名的本质用来表示函数首地址—函数第一条指令的地址。一个函数指针指向函数后，指针变量的值等于所指向函数的首地址。
- 如果在程序中输出函数指针值和指针所指向的内容，输出的内容都是函数的首地址。这种指针和指针指向的内容是一致的情况，仅出现在函数指针和数组的指针中。



函数指针的意义

- 函数指针的最大用途就是通过函数指针调用一个函数。
- 使用函数指针调用函数和使用函数名调用函数有什么本质的区别呢？
- 其本质区别在于函数指针可以在程序运行的过程中动态地决定调用哪一个函数，而是用函数名调用函数的话就将调用的函数写死了。



函数指针作为函数返回值

```
void (*f(int a, int b))(int);
```

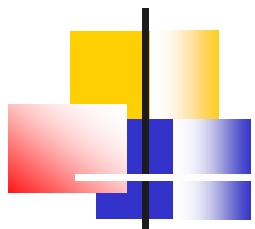
这个声明代表什么？

它代表声明一个函数**f**，它的两个参数是两个整形，它的返回值是一个函数指针，这个函数指针指向的函数的返回值是**void**，而它的参数是整形。

通常来说对于复杂的类型定义会用**typedef**进行描述，以方便阅读，以上函数**f**声明也可以写成：

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

```
func_p f(int a, int b);
```



函数指针作为函数返回值

- 分析该声明主要注意：
 - **f**是一个函数名，不是函数的指针
 - **f**的返回值是函数的指针，而不是**void**
 - 对于函数**f**来说，参数是两个整形，对于**f**的返回值来说（这个返回值是一个函数的指针），它的参数是一个整形

综合实例

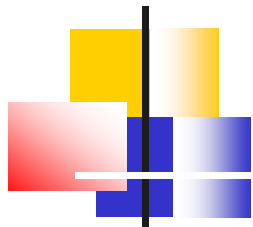
■ 练习

通过函数指针调用函数

```
typedef int (*funcp_t)(int, int);  
int add(int a, int b){return a + b;}
```

Section3

函数指针数组



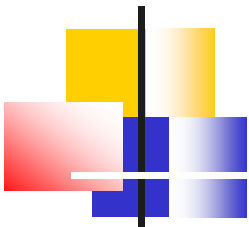
函数指针的数组定义

- 一般的定义形式：

函数返回值类型 （*数组名[长度]） （参数列表）；

如：

```
int (*f[10])(int, int);
```

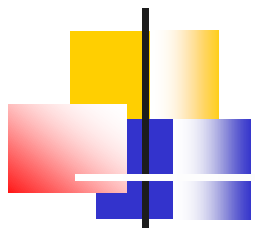



函数指针数组

```
int (*f[10])(int, int);
```

定义了一个函数指针数组**f**，该数组的每一个元素**f[i]**用于存放一个函数指针，指向的函数为拥有两个整形参数、一个整形返回值的形式。

问：**f**占多少字节？



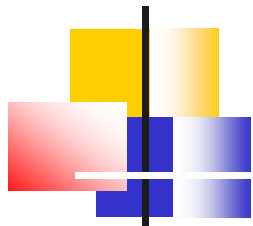
函数指针数组

■ 实例

有两个函数

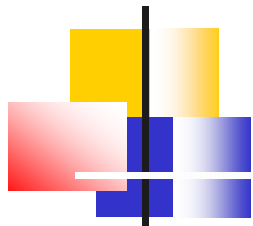
```
int add(int a, int b){return a + b;}
```

```
int mul(int a, int b){return a * b;}
```



函数指针数组

```
int main(void)
{
    int (*op[2])(int, int);
    op[0] = add;
    op[1] = mul;
    printf("%d %d\n", op[0](1, 2), op[1](1, 2));
    return 0;
}
```



函数指针数组

■ 几种容易混淆的写法

```
int *p[10];
```

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

```
int (*p)(int);
```

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

函数指针数组

- 练习
- 使用函数指针数组实现一个小型计算器，计算器支持加减乘除四则运算。

如输入：+

2 3

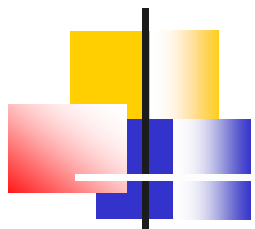
输出结论为：5

输入：'#'，表示计算器退出



Section4

回调函数

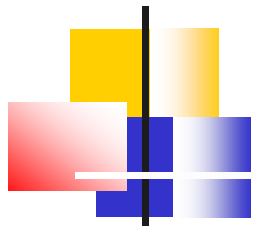


回调函数

- 函数指针作为函数的参数是一种很重要的用法，根据参数可以选择调用不同的函数。

```
typedef void (*func_p)(void);  
void func(func_p op);
```

以上定义了一个函数**func**，其参数为一个函数指针。

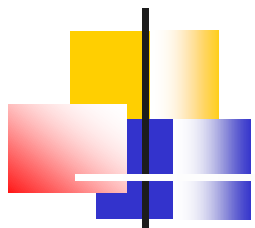


回调函数

```
void func(func_p op);
```

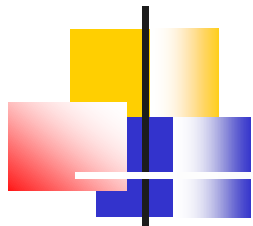
像这种函数，参数为一个函数指针，即调用者提供一个函数但自己不去调用它，而是让实现者去调用它，这称为回调函数。

回调函数中应该会有语句进行函数调用



使用回调函数实现泛型算法

- 回调函数的一个重要应用实例就是实现“泛型”算法。泛型算法是一种算法实现的方法，这种实现方法的优势着重体现在对数据类型的普适性上，即：
对同一类问题，要操作的数据类型可能不同，但解决办法应该相同



泛型函数

- 泛型算法在实现的过程中解决算法的大部分流程，只留下少量的处理细节的代码给使用泛型函数的用户去完成。
- 泛型算法的指导思想就是代码的复用。

泛型函数

■ 练习

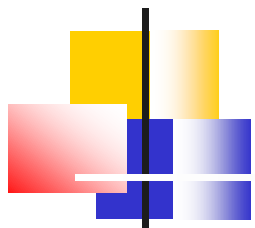
■ 有如下需求

从一组整形数据中找出最大值

从一组字符串中找出字典序最大的字符串

从一组学生的信息中找到成绩最高的学生

要求使用泛型函数解决这类问题，对以上三种情况均适用



泛型函数

- 使用泛型函数实现冒泡排序，要求
 - 整形数据
 - 字符串
 - 学生的信息（按照成绩）
 - 对以上类型的数据进行排序

总结

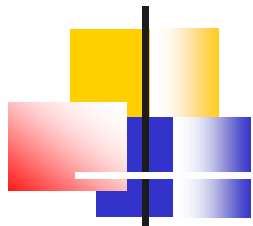
■ 函数指针的基本概念

- 定义和引用一个函数指针
- 函数指针作为函数返回值
- 函数指针作为函数参数

■ 函数指针数组

- 函数指针数组的定义和引用
- C语言中容易混淆的指针定义

■ 回调函数



Let's DO it!

Thanks for listening!

