C语言程序设计基础

林川

第10章 程序结构

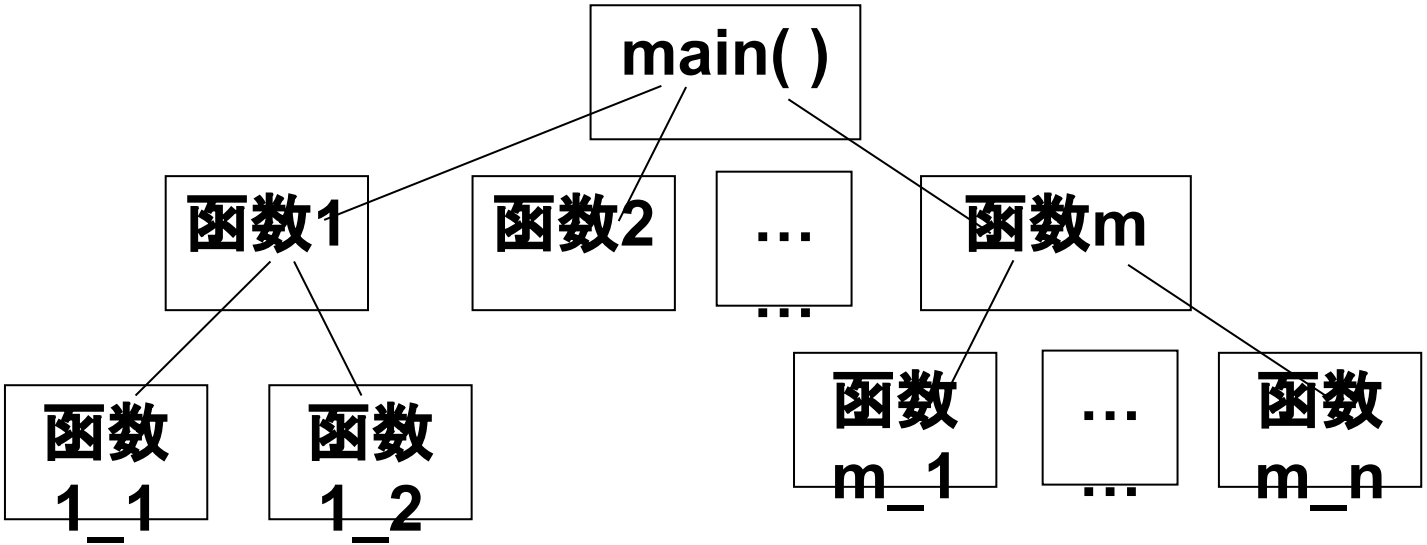
本章要点

- 怎样把多个函数组织起来？
- 怎样用结构化程序设计的思想解决问题？
- 怎样用函数嵌套求解复杂的问题？
- 怎样用函数递归解决问题？
- 如何使用宏？

结构化程序设计

- 使用结构化程序设计方法解决复杂的问题
  - 把大问题分解成若干小问题，小问题再进一步分解成若干更小的问题
  - 写程序时，用main()解决整个问题，它调用解决小问题的函数
  - 这些函数又进一步调用解决更小问题的函数，从而形成函数的嵌套调用

程序结构

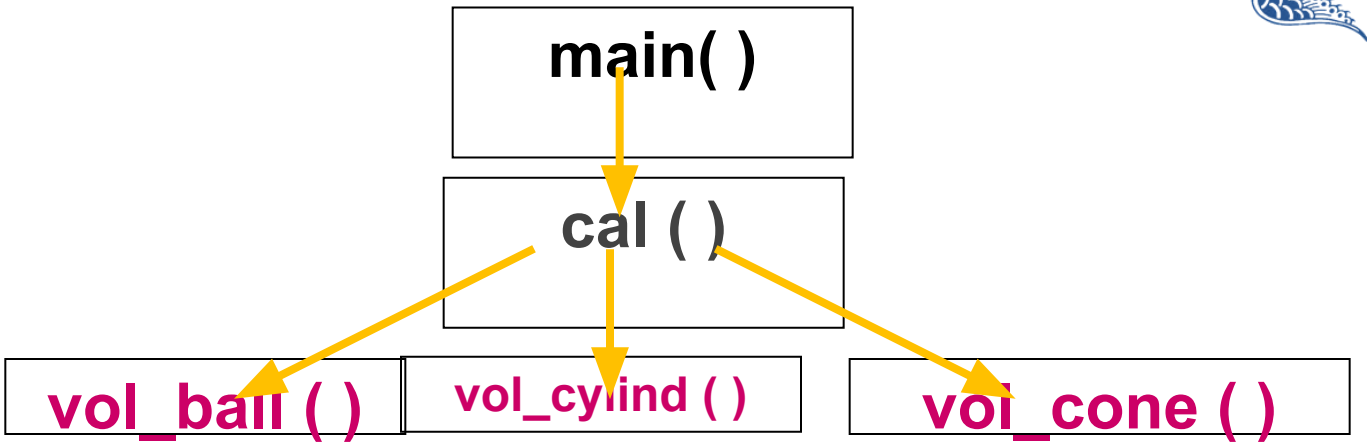


[例程10-1] 计算常用圆形体体积

设计一个常用圆形体体积计算器，采用命令方式输入1、2、3，分别选择计算球体、圆柱体、圆锥体的体积，并输入计算所需相应参数。

- 分析：
  - 输入1、2、3选择计算3种体积，其他输入结束计算
  - 设计一个控制函数cal()，经它辨别圆形体的类型再调用计算球体、圆柱体、圆锥体体积的函数
  - 设计单独的函数计算不同圆形体的体积

程序结构



- 3层结构，5个函数
- 降低程序的构思、编写、调试的复杂度
- 可读性好

例10-1源程序

```
#define  PI  3.14159265
void cal ( int sel );
int main(void)
{
    int sel;
    while( 1 ){
      printf("  1-计算球体体积\n");
      printf("  2-计算圆柱体积\n");
      printf("  3-计算圆锥体积\n");
      printf("  其他 -退出运行\n");
      printf("请输入计算命令:");
      scanf("%d",&sel);
      if (sel<1 || sel>3)
        break
      else
        cal(sel);
      }
      return 0;
    }
```

主控函数 cal

```
void cal ( int sel )
{
   double  vol_ball(void );
   double  vol_cylind(void );
   double  vol_cone(void );

   switch (sel) {
      case 1: printf("球体积为:%.2f\n", vol_ball( ));
              break;
      case 2: printf("圆柱体积为:%.2f\n", vol_cylind( ) );
              break;
      case 3: printf("圆锥体积为:%.2f\n", vol_cone( ) );
              break;
   }
}
```

球体体积函数vol_ball

```
double vol_ball( )
{
    double r ;
    printf("请输入球的半径:");
    scanf("%lf",&r);
    return(4.0/3.0*PI*r*r*r);
}
```

圆柱体积函数vol_cylind( )

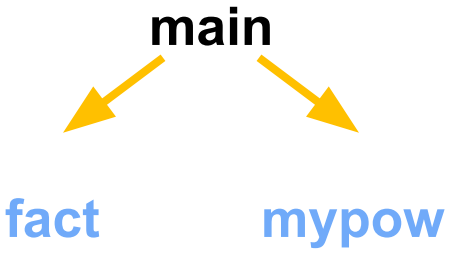
```
double vol_cylind( )
{
    double r , h ;
    printf("请输入圆柱的底圆半径和高 : ");
    scanf("%lf%lf",&r,&h);
    return(Pl*r*r*h);
}
```

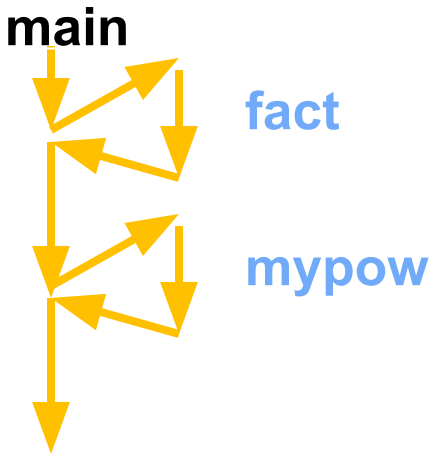
圆锥体积函数vol_cone

```
double vol_cone( )
{
    double r , h ;
    printf("请输入圆锥的底圆半径和高 : ");
    scanf("%lf%lf",&r,&h);
    return(PI*r*r*h/3.0);
}
```

10.1.2 函数的嵌套调用

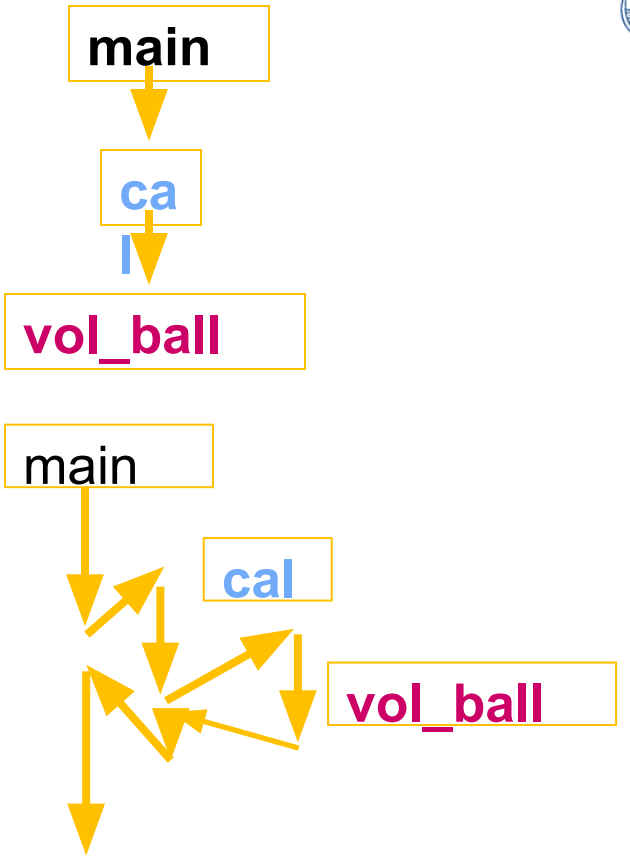
```
int main(void)
{
    ......
    y = fact(3);
    ......
    z = mypow(3.5, 2);
    ......
}
double fact(int n)
{
     ......
}
double mypow(double x, in n)
{
     ......
}
```





函数的嵌套调用

```
int main(void)
{   ......
    cal (sel);
    ......
}
void cal (int sel)
{       ......
    vol_ball()
    ......
}
double vol_ball( )
{
    ......
}
```



函数的嵌套调用

- 在一个函数中再调用其它函数的情况称为函数的嵌套调用。
- 如果函数A调用函数B，函数B再调用函数C，一个调用一个地嵌套下去，构成了函数的嵌套调用。
- 具有嵌套调用函数的程序，需要分别定义多个不同的函数体，完成不同的功能，它们合起来解决复杂的问题。
- 循环嵌套调用
    - 递归函数，递归调用

10.2.2 递归函数基本概念

阶乘的递归定义：

n! = n * (n-1)!, 当n>1时；

n! = 1，当n=0或n=1时。

□递归通式

```
double  fact( int n )
{
  double result;
  if( n<=1 )
     result = 1.0;
  else
     result = n * fact(n-1);
  return result;
}
```

□递归出口

函数fact自己调用自己
称作：递归调用

不能写成：
fact(n) = n*fact(n-1);

10.2.2 递归函数基本概念

直接递归调用

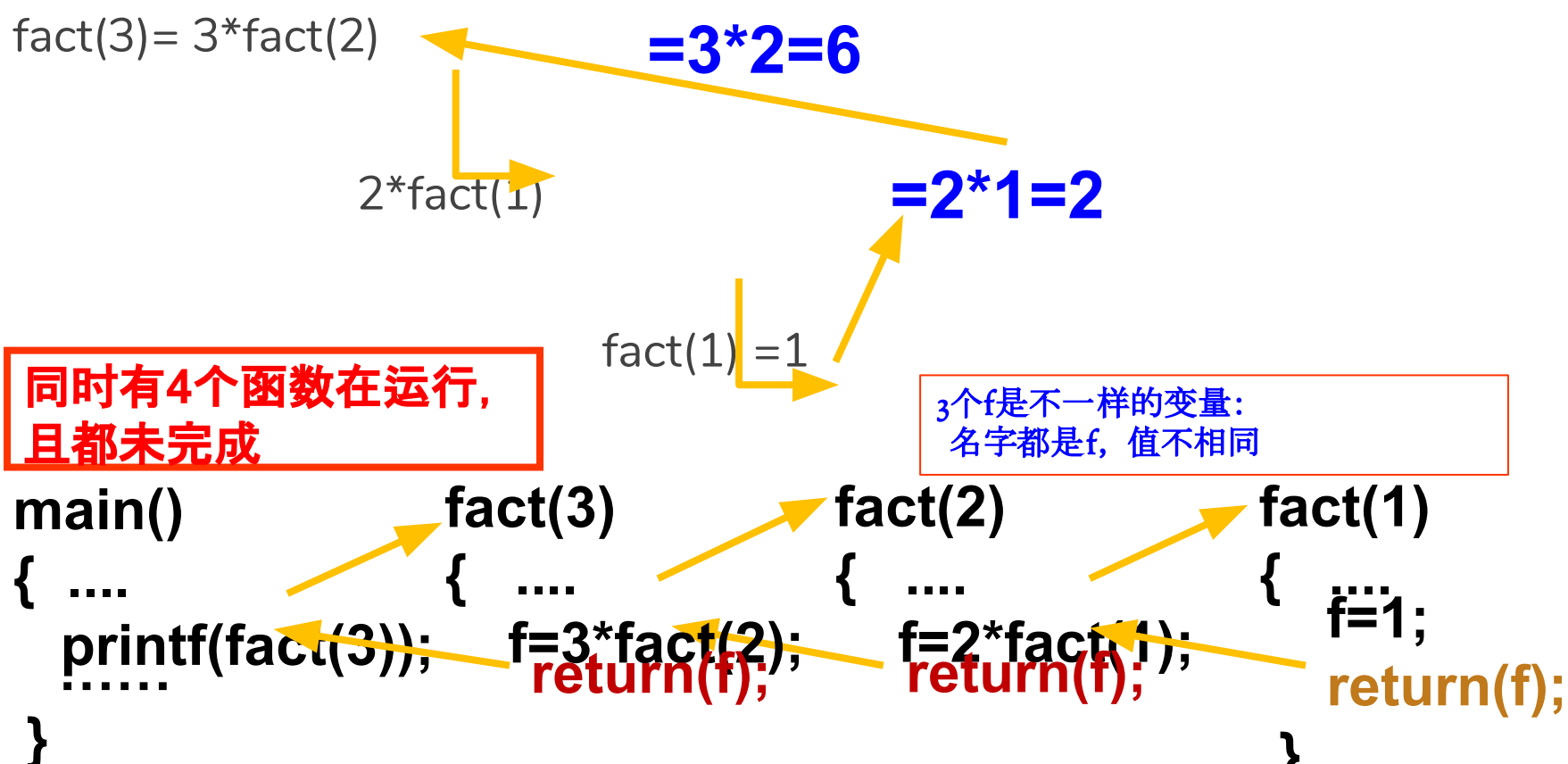
```
int f(int x)
{
   …
   y = f(x-1)
   …
   return y;
}
```

间接接递归调用

```
int f(int x)
{
   …
   y = g(x-1)
   …
   return y;
}
```

```
int g(int x)
{
   …
   z = f(x-1)
   …
   return z;
}
```

递归函数 fact(n)的实现过程



例程10-3 将整数按照逆序输出

```
void reverse(int num )
{
   printf("%d", num%10);
   if( num>9 )
      reverse( num/10 );

}
```

执行reverse(12345)

输出：5

执行reverse(1234)

输出：4

执行reverse(123)

输出：3

执行reverse(12)

输出：2

执行reverse(1)

输出：1

因此，最终输出为 54321

递归程序设计

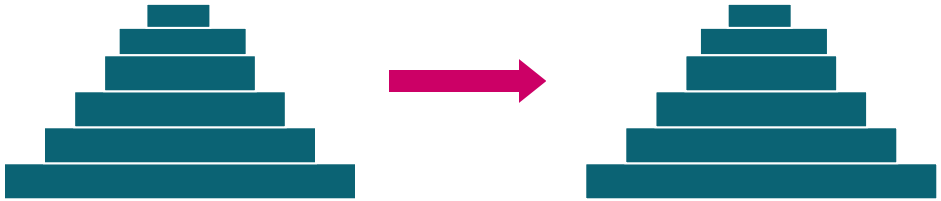
用递归实现的问题，满足两个条件：

- 问题可以逐步简化成自身较简单的形式（递归式）
  - 参数逐渐减小
- 递归最终能结束(递归出口)

两个条件缺一不可

解决递归问题的两个着眼点

例10-5 汉诺(Hanoi)塔



A B C

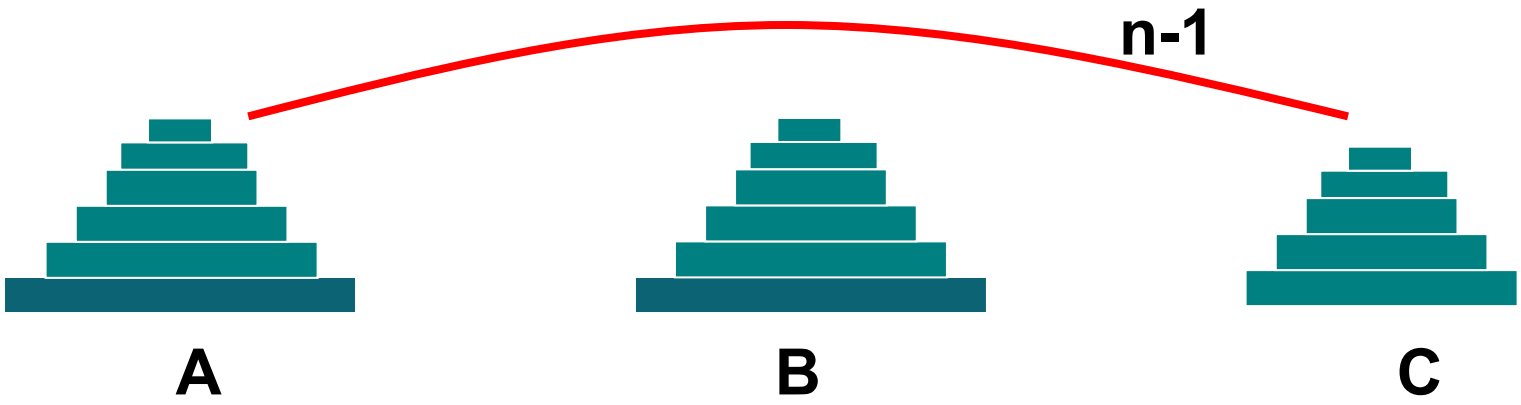
将64 个盘从座A搬到座B

(1) 一次只能搬一个盘子

(2) 盘子只能插在A、B、C三个杆中

(3) 大盘不能压在小盘上

分析



汉诺(Hanoi)塔 问题求解

伪代码描述

```
hanio(n个盘, A→B, C为过渡)
{
  if (n == 1)
    直接搬运盘子:A→B
   else
   {
     hanio(n-1个盘, A→C,
         B为过渡)
      搬运第n个盘子:A→B
      hanio(n-1个盘, C→B,
         A为过渡)
   }
}
```

C语言代码

```
void
hanio(int n, char A, char B, char C)
{
 if (n == 1)
   printf("%c-->%c\n", A, B);
 else
 {
   hanio(n-1, A, C,
        B);
   printf("%c-->%c\n", A, B);
   hanio(n-1, C, B,
        A);
 }
}
```

10.3 宏定义

- #define 宏名标识符 宏定义字符串

  #define PI 3.14

  #define arr_size 4

  编译时，把程序中所有与宏名相同的标识符，用宏定义字符串替代

说明:

- 宏名一般用大写字母，以与变量名区别
- 宏定义不是C语句，后面不得跟分号
- 宏定义可以嵌套使用
- 多用于符号常量、简单的操作和函数等

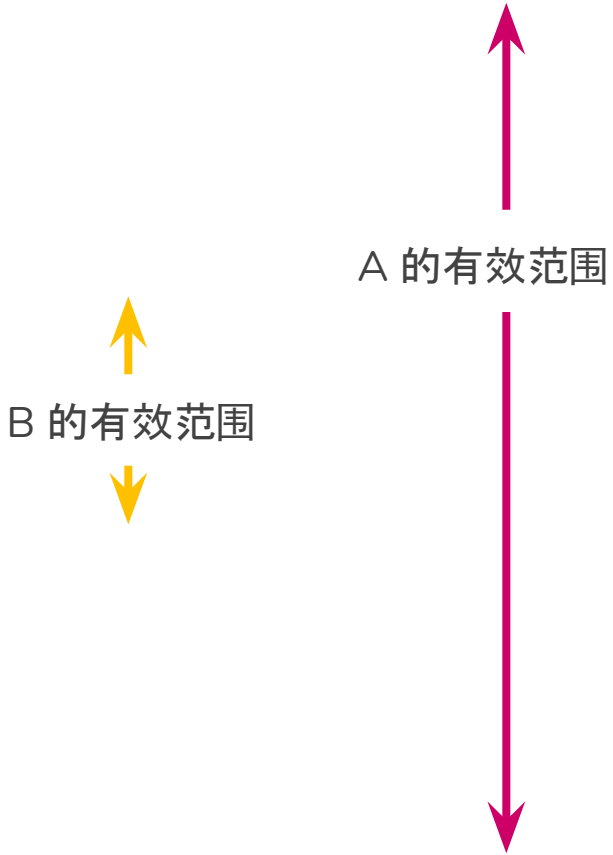
  #define MAX(a, b) ((a)>(b) ? (a) : (b))

10.3.1 宏基本定义

- 宏定义可以写在程序中任何位置，它的作用范围从定义书写处到文件尾。
- 可以通过“#undef”强制指定宏的结束范围。

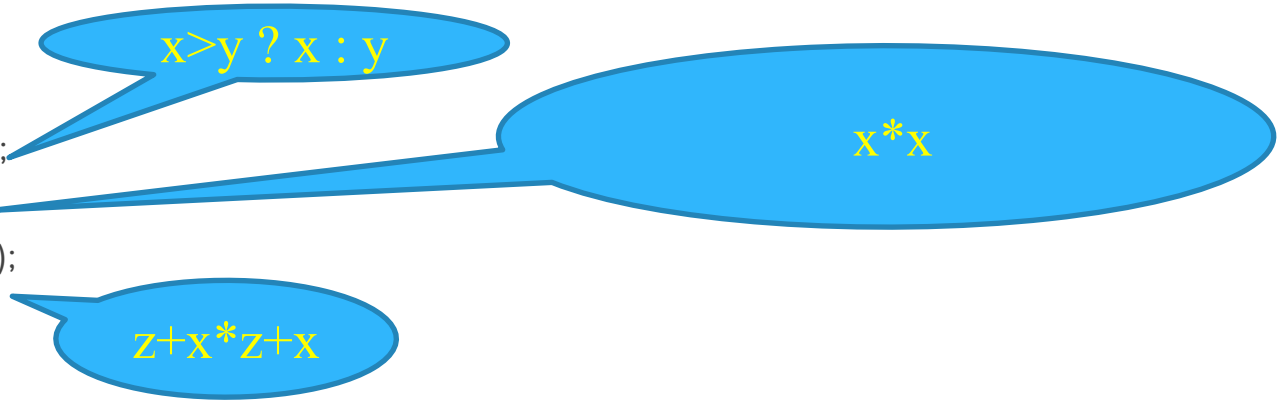
宏的作用范围

```
#define  A  “This is the first macro”
void  f1()
{
   printf( “A\n” );
}
#define  B  “This is the second macro”
void  f2( )
{
   printf( B ) ;
}
#undef  B
int main(void)
{
   f1( );
   f2( );
   return 0;
}
```



10.3.2 带参数的宏定义

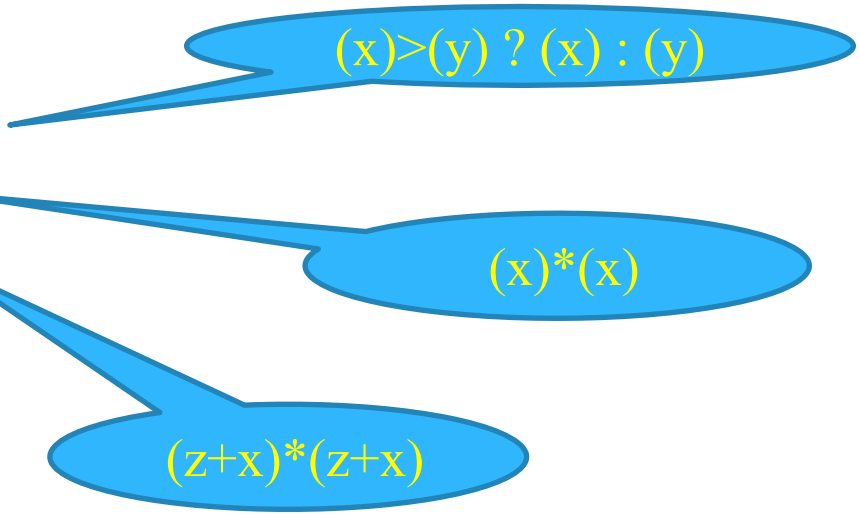
```
#define MAX(a,b)   a>b ? a : b
#define SQR(x)     x*x
...
int x, y, z
...
x = MAX(x,y);
y = SQR(x);
y = SQR(z+x);
```



10.3.2 带参数的宏定义

建议使用()，减少麻烦

```
#define MAX(a,b)   (a)>(b) ? (a) : (b)
#define SQR(x)     (x)*(x)
...
int x, y, z
...
x = MAX(x,y);
y = SQR(x);
y = SQR(z+x);
```



用宏实现两个变量的交换

```
#define  f(a, b, t) (t)=(a),(a)=(b),(b)=(t)
void main( void )
{
  int x,y,t ;
  s (t)=(x),(x)=(y),(y)=(t);;;
  f(x,y,t);/*使用宏*/
  printf("%d  %d\n", x, y) ;
```

嵌套的宏定义与调用

```
#define F(x)  x - 2
#define D(x)  x*F(x)
void main()
{
   printf("%d,%d", D(3), D(D(3))) ;
}
```

嵌套的宏定义

计算 D(3) 和 D(D(3))

```
#define F(x)  x - 2
#define D(x)  x*F(x)
```

- 先全部替换好，最后再统一计算

D(3) = 3*F(3)
= 3*3-2
= 7

D(D(3)) = D(3)*F(D(3))
= 3*F(3)*D(3)-2
= 3*3-2*3*F(3)-2
= 3*3-2*3*3-2-2
= -13

运行结果：7，-13

宏定义应用示例

- 判断字符c是否为小写字母。

```
#define  LOWCASE(c)  (((c) >= 'a') && ((c) <= 'z') )
```

- 将数字字符（‘0’～‘9’）转换为相应的十进制整数，-1表示出错。

```
#define  CTOD(c)  (((c) >= '0') && ((c) <= '9') ? c - '0' : -1)
```

- 最大值、最小值

```
#define  MAX(a,b)  ((a) >= (b) ?  (a) : (b))
#define  MIN(a,b)  ((a) <= (b) ?  (a) : (b))
```

10.3.4 文件包含

- 为了避免一个文件过长，可以把程序分别保存为几个文件。
- 一个大程序会由几个文件组成，每一个文件又可能包含若干个函数。
- 程序文件□程序文件模块。
  - 程序 □ 文件 □ 函数
- 大程序—若干程序文件模块
  - 各程序文件模块分别编译，再连接
- 整个程序只允许有一个main()函数

文件包含

- 格式

  #include <需包含的文件名>

  #include "需包含的文件名"

例如

#include <stdio.h>

#include "myfunc.h"

- 作用
  - 把被包含的文件内容插入到#include命令所在位置

注意

- 编译预处理命令，以#开头。
- 行尾没有分号。

[例10-7] 文件包含举例

- 文件length.h

```
/* 1英里=1609米 */
#define mile_to_meter 1609
/* 1英尺=30.48厘米 */
#define foot_to_centimeter 30.48
/* 1英寸=2.54厘米 */
#define inch_to_centimeter 2.54
```

[例10-7] 文件包含举例

- 文件 prog.c

```
#include <stdio.h>
#include “length.h”
void main()
{
   float foot, inch, mile;
   printf(“%f miles = %f\n”,
       mile, mile*mile_to_meter);
   …
}
```

.h头文件常规用法

- 统一的定义和声明
  - 宏定义
  - 变量声明
  - 函数申明
  - 外部变量申明
  - 自定义的数据类型(结构)
- 避免多次重复定义
- 避免不一致
- 方便修改

常用标准头文件

- ctype.h 字符处理
- math.h 与数学处理函数有关的说明与定义
- stdio.h 输入输出函数中使用的有关说明和定义
- string.h 字符串函数的有关说明和定义
- stddef.h 定义某些常用内容
- stdlib.h 杂项说明
- time.h 支持系统时间函数

10.3.5 编译预处理

- 编译预处理是C语言编译程序的组成部分，它用于解释处理C语言源程序中的各种预处理指令。
- 文件包含(#include)和宏定义(#define)都是编译预处理指令
  - 在形式上都以“#”开头，不属于C语言中真正的语句
  - 增强了C语言的编程功能，改进C语言程序设计环境，提高编程效率

编译预处理

- 由于#define等编译预处理指令不是C语句，不能被编译程序翻译
- 需要在真正编译之前作一个预处理，解释完成编译预处理指令
- 从而把预处理指令转换成相应的C程序段，最终成为由纯粹C语句构成的程序
- 经编译最后得到目标代码。

编译预处理功能

- 编译预处理的主要功能：
  - 文件包含（#include）
  - 宏定义（#define）
  - 条件编译

条件编译

```
#define _flag 1
#define _flag 0
#if _flag
程序段1
#else
程序段2
#endif
```

条件编译

```
#define _zhongwen
#ifdef _zhongwen
#define msg “早上好”
#else
#define msg “Good morning”
#endif
...
main()
{
   printf(msg);
}
```

条件编译(用于调试)

```
#define _mydebug 0
#if _mydebug
#define _dbg(x) (x)
#else
#define _dbg(x)
#endif
```

```
main()
{
   int x;
   ...
   _dbg(printf("%d",x);
   ...
}
```

10.4 大程序构成 - 多文件模块

- **学生信息库系统**
  - **建立 new_student()**
  - **输出 output_student()**
  - **计算平均成绩 average()**
  - **平均成绩排序 sort()**
  - **修改 modify()**
  - **查询 search_student()**

- 文件模块

  student.h

  input_output.c

  aver_sort.c

  modify.c

  student_system.c

大程序构成 - 多文件模块

- 用户头文件 - student.h
  - 宏定义，数据类型定义(结构)

```
#include<stdio.h>
#include<string.h>
#define MaxSize 50
struct student {
  int   num;
  char  name[10];
  int   computer, english, math;
  float average;
};
```

大程序构成 - 多文件模块

- 用户头文件 - student.h《续》
  - 外部变量，外部函数

```
extern int count;
void new_student(struct student students[]);
void output_student(struct student students[]);
void average(struct student students[]);
void sort(struct student students[]);
void modify(struct student students[]);
void search_student(struct student students[],
                    int num);
```

大程序构成 - 多文件模块

输入/出程序文件 – input_output.c

```
#include “student.h”
void new_student(struct student students[])
{
  …
}

void output_student(struct student students[])
{
  …
}
```

大程序构成 - 多文件模块

- 计算平均成绩及排序文件 – aver_sort.c

```
#include “student.h”
void average(struct student students[])
{
  …
}

void sort(struct student students[])
{
  …
}
```

大程序构成 - 多文件模块

- 查询和修改文件 – modify.c

```
#include “student.h”
void modify(struct student students[])
{
  …
}

void search_student(struct student students[],
                    int num)
{
  …
}
```

大程序构成 - 多文件模块

- 主函数程序– student_system.c

```
#include “student.h”
int count = 0;
int main()
{
  struct student students[MaxSize];

  … /* 调用函数，实现功能*/
}
```

10.4.3 文件模块之间的沟通

- 外部变量
  - 在文件A中使用文件B中定义的全局变量c的时候，需要在使用之前进行声明
- 声明格式如下：

  extern 类型名 变量名;

  例如:

  extern int count;
- 如果不希望其他文件使用某个全局变量，那么需要将其定义为static类型的全局变量

10.4.3 文件模块之间的沟通

- 外部函数
  - 在文件A中使用文件B中定义的函数f的时候，需要在使用之前进行声明
- 声明格式如下：

extern 类型名 函数名(参数类型表);

关键字 extern可以省略

例如:

```
void new_student(struct student students[]);
void output_student(struct student
                    students[]);
```

10.4.3 文件模块之间的沟通

- 静态函数
  - 如果不希望文件中的某个函数在其他的文件中调用，那么可以将其定义为静态的
  - 也称作：内部函数

声明格式如下：

```
static 类型名 函数名(参数类型表)
{
   …
}
```