

第 6 讲

模块化程序设计： 函数

主要内容

- 6.1 模块化程序设计概述
- 6.2 函数的定义与声明
- 6.3 函数的调用与返回值
- 6.4 函数的作用域与生存期
- 6.5 模块化程序设计举例

6.1 模块化程序设计概述

- 模块化程序设计是指以“**模块**”为中心的
程序设计过程
- 一般程序设计语言提供了一种机制——**子
程序技术**，一个子程序就是一个程序模块
- 在C中，子程序体现为函数

● 子程序

- ✓ subroutine, procedure, function, subprogram, routine
- ✓ 在程序设计过程中，如果其中有些操作内容完全相同或相似，为了简化程序，可以把这些重复的程序段单独列出，并按一定的格式编写成子程序。子程序负责完成某项特定任务，而且相较于其他代码，具备相对的独立性
- ✓ 主程序在执行过程中如果需要某一子程序，通过调用指令来调用该子程序，子程序执行完后又返回到主程序，继续执行后面的程序段

- 子程序技术是“自顶向下、逐步求精”设计策略的**基础**，它将**what**和**how**分离开
 - ✓ 设计时，先按照自顶向下策略，为完成某个操作，程序员只需关心要“**做什么**”（调用一个子程序）即可，而不必关心“怎么做”，从而可以集中精力处理好高层架构
 - ✓ 然后在逐步求精过程中，再去处理“**怎么做**”（设计子程序本身），设计具体的算法细节

- 模块化程序设计采用“**组装**”的办法简化程序设计过程
 - ✓ 将重复的处理放在一个函数中，只需编写一次，从而可以实现代码的复用
 - ✓ 允许程序员将实现细节隐藏在模块内部。他人如果只是从功能的角度使用模块，可忽略细节
 - ✓ 每个模块独立开发和测试，有利于信息隐藏，以及团队分工合作并行开发
 - ✓ 模块作为整体方案的一小部分，调试维护方便
 - ✓ 使用模块，程序逻辑结构清晰、易写易读易懂

- 模块化程序设计的主要内容
 - ◆ 模块的分解
 - ◆ 模块间语句的划分
 - ◆ 模块间信息的传递

模块的分解

- 模块分解的原则
 - ✓ 一般按功能分解，保持模块的**内聚性**（独立性），使一个模块完成一项功能
 - ✓ 减少模块的**耦合性**，保证模块间的信息相对封闭
 - ✓ **模块数**适中。随着模块数的增加，模块间的接口设计的复杂性和工作量将随之增加

- 模块分解方法

- ✓ 自顶向下：步步深入、逐层细分

- 先整体后局部、先抽象后具体

- 主要用于从无到有设计一个程序

- ✓ 自底向上：由表及里、由浅入深

- 先底层后顶层、先分散后集成

- 主要用于修改、优化或扩充程序

模块间语句的划分

例：编写一个程序，实现从键盘上输入两个整数，输出其中的较大值

- ✓ 要求程序中含有一个主函数、一个子函数
- ✓ 子函数的主要功能是比较两个数
- ✓ 主函数调用子函数

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

```
void f1( )    // 包含 I、P、O
```

方法一：

```
{ int x, y, z;
```

```
    scanf("%d%d", &x, &y);
```

```
    if(x>y) z=x;
```

```
    else z=y;
```

```
    printf("The big is %d!\n", z);
```

```
}
```

```
void main( )
```

```
{
```

```
    f1();
```

```
}
```

```
#include<stdio.h>
void f2(int a, int b) // 包含 P、O
{
    int c;
    if(a>b) c=a;
    else c=b;
    printf("The big is %d!\n", c);
}
void main( ) // 包含 I
{
    int x, y;
    scanf("%d%d", &x, &y);
    f2(x, y);
}
```

方法二：

```
#include<stdio.h>
int f3(int a, int b)    // 包含 P
{
    int c;
    if(a>b) c=a;
    else c=b;
    return c;
}
void main( )            // 包含 I、O
{
    int x, y, z;
    scanf("%d%d", &x, &y);
    z=f3(x, y);
    printf("The big is %d!\n", z);
}
```

方法三：

- 模块间语句的划分的一般原则

- ✓ 方法一大包大揽，对于要求所有输入输出和处理均在模块内完成的情形，采用方法一
- ✓ 对于要求一边处理一边输出结果的情形，采用方法二
- ✓ 对于功能较为复杂、且需要多次调用的情形，采用方法三，即将输入、输出操作放到主调函数中，数据处理操作放到子函数中
- ✓ 如果输入输出操作比较复杂，可以将输入、输出和处理定义为独立的子函数

模块间信息的传递

- 模块间的信息交换

- ✓ 主调模块将信息发送给被调用模块，接收来自被调用模块的处理结果
- ✓ 被调用模块接收来自主调模块的信息，处理完成后将结果返回给主调模块
- ✓ 信息传递的方式有值传递和地址传递等
- ✓ 一般采用自顶向下方法，将输入、处理、输出设计成独立模块，模块间相互交换信息

● C语言中的程序模块——函数

- ◆ 在C中，程序的每个模块是一个函数
- ◆ 经常使用的模块通常都是以函数库的形式存在于计算机系统中，需要时调用即可
- ◆ 程序较小时，通常只有一个函数；当程序较大时，往往将部分功能相对独立的语句组织成若干个函数

◆ C语言中函数的分类

- ✓ 按功能分：库函数、自定义函数
- ✓ 按形式分：有参函数、无参函数
- ✓ 按有无返回值分：普通函数、过程函数
- ✓ 按身份分：主函数、子函数
- ✓ 按承担角色分：主调函数、被调函数

6.2 函数的定义与声明

- “函数定义”要解决的主要问题：
 - ✓ 函数名是什么？
 - ✓ 函数的参数个数及其类型是什么？
 - ✓ 函数返回值的类型是什么？
 - ✓ 函数要实现的功能是什么？

返回值类型 函数名(参数列表)

{

函数体

}

无参函数的定义

指定函数值的类型，可以是
int、float、double、char、
void、指针或结构体类型

返回值类型 函数名()

{

函数体

}

返回值类型 函数名(void)

{

函数体

}

包括声明部分和语
句部分

有参函数的定义：

```
返回值类型 函数名(类型 参数1, 类型 参数2, ..., 类型 参数n)
{
    函数体
}
```

说明:

- (1) 函数返回值（一般通过被调用函数中的return语句带回）的类型，简称**函数类型**，缺省的函数类型为int，不允许是数组或函数类型；当函数无返回值时，函数类型为void
- (2) 函数名符合标识符的命名规则
- (3) 参数表列中包括**0~多个**参数（称为“形式参数”，简称**形参**），它是主调函数与被调函数间信息的传递接口，应分别给出各参数的类型，参数之间以“,”相隔

- (4) 函数体由局部变量定义（声明）和语句组成。语句部分规定了在本函数中要执行的操作（功能）
- (5) 函数一经定义，将在编译阶段由系统给该函数在内存中分配一段连续的空间，在该空间存储一系列的程序指令等。**这段空间的首地址由函数名标识**，即**函数名代表了函数在内存中的首地址**

例1:

某加油站当前油价如下：98#汽油8.23元/升，95#汽油7.89元/升，92#汽油7.35元/升。为提高效率，加油站推出了“自助服务（Z）”和“协助服务（X）”两种服务，可分别得到3%和1%的折扣。请编写程序，当从键盘上输入汽油种类x、加油量y和服务类型z，计算并输出应付款额cost（保留小数点后面2位）


```
#include "stdio.h"
```

```
void main() {
```

```
    int x;    char z;    float y, dis, cost;
```

```
    FLAG:
```

```
    printf("请输入汽油种类x(92/95/98)、加油量y和服务类型z(Z/X):\n");
```

```
    scanf("%d,%f,%c", &x, &y, &z);
```

```
    if(z=='Z' || z=='z') dis=0.97;
```

```
    else if(z=='X' || z=='x') dis=0.99;
```

```
        else { printf("Input error.\n"); goto FLAG; }
```

```
    switch(x) {
```

```
        case 92: cost=7.35*y*dis; break;
```

```
        case 95: cost=7.89*y*dis; break;
```

```
        case 98: cost=8.23*y*dis; break;
```

```
        default: printf("Oil type error.\n"); goto FLAG;
```

```
    }
```

```
    printf("Total fee is %.2f元.\n", cost);
```

```
}
```

方法一：

```
请输入汽油种类x(92/95/98)、加油量y和服务类型z(Z/X):
95, 50, Z
Total fee is 382.67元.
```

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

```
void main( )
```

方法二：

```
{
```

```
    int x;      char z;      float y, cost;
```

```
    FLAG:
```

```
    printf("请输入汽油种类x(92/95/98)、加油量y和服务类型  
          z(Z/X):\n");
```

```
    scanf("%d,%f,%c", &x, &y, &z);
```

```
    if(x!=92&&x!=95&&x!=98) {printf("Oil type error.\n"); goto FLAG;}
    if(z!='Z'&&z!='z'&&z!='X'&&z!='x')
```

```
        { printf("Input error.\n"); goto FLAG; }
```

```
    cost=Oil_fun(x, y, z);
```

```
    printf("Total fee is %.2f元.\n", cost);
```

```
}
```

```

float Oil_fun(int a, float b, char c) // 函数头部
{
    float fee, dis; // 定义函数中用到的局部变量
    if(c=='Z' || c=='z') dis=0.97;
    else if(c=='X' || c=='x') dis=0.99;
    switch(a)
    {
        case 92: fee=7.35*b*dis; break;
        case 95: fee=7.89*b*dis; break;
        case 98: fee=8.23*b*dis; break;
    }
    return fee;
}

```

error C2065: 'Oil_fun' : undeclared identifier

- “函数声明”：

- ✓ 函数须先定义后调用
- ✓ 如果先调用后定义，则需要进行声明
- ✓ C中引进了“函数原型”(Prototype)的概念
- ✓ 函数原型是一个声明，放在函数调用之前，先声明相应函数的特性，向编译系统提供准确的函数接口信息，从而满足了先定义后调用的要求

无参函数的声明：

返回值类型 函数名();

返回值类型 函数名(void);

有参函数的声明：

返回值类型 函数名(类型 参数1, 类型 参数2, ..., 类型 参数n);

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

```
void main( )
```

```
{
```

```
    float Oil_fun(int a, float b, char c);
```

```
    int x;      char z;      float y, cost;
```

```
    FLAG:
```

```
    printf("请输入汽油种类x(92/95/98)、加油量y和服务类型(Z/X):\n");
```

```
    scanf("%d,%f,%c", &x, &y, &z);
```

```
    if(x!=92&&x!=95&&x!=98) {printf("Oil type error.\n"); goto FLAG;}
    if(z!='Z'&&z!='z'&&z!='X'&&z!='x')
```

```
        { printf("Input error.\n"); goto FLAG; }
```

```
    cost=Oil_fun(x, y, z);
```

```
    printf("Total fee is %.2f元.\n
```

```
}
```

方法二（修改）：

```
请输入汽油种类x(92/95/98)、加油量y和服务类型(Z/X):
95, 50, Z
Total fee is 382.67元.
```

说明:

- (1) 函数声明不是创建新的函数和功能，而是对在后面定义的函数的补充说明，是为了获得调用函数的权限，方便调用和保护源代码
- (2) 函数原型中，参数表列中的参数名不起作用。因此，只需说明参数个数和每个参数的类型。
也可：`float Oil_fun(int, float, char);`
- (3) 程序中需调用的所有函数的原型一般放在程序的开头，或主调函数的函数体的最前面
- (4) .h文件中存放了相关的系统函数的原型

6.3 函数的调用与返回值

- 程序的入口点是主函数，其他函数的运行是靠别的函数调用启动的
- 函数调用的格式：

函数名()

或 函数名(实际参数表)

- 函数调用的形式：

(1) 作为表达式的一部分参与运算

如： `cost=Oil_fun(x, y, z);`

(2) 函数调用语句

如： `printf("Total fee is %.2f元.\n", cost);`

(3) 作为函数参数

如： `printf("Total fee is %.2f元.\n", Oil_fun(x, y, z));`

- 函数调用的执行过程：

- (1) 函数的嵌套调用

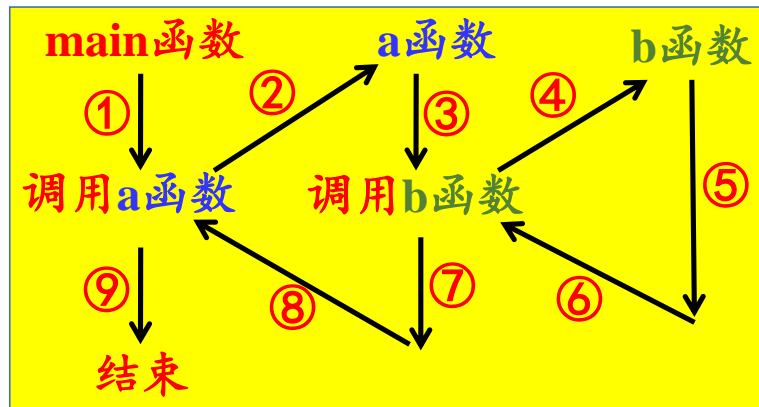
- (2) 函数的递归调用

- ✓ 函数的直接递归调用

- ✓ 函数的间接递归调用

(1) 函数的嵌套调用

```
#include<stdio.h>
void main( ){
    void a(), b();
    printf("这是第1个输出! \n");
    a();
    printf("这是第5个输出! \n");
}
void a(void)
{
    void b();
    printf("这是第2个输出! \n");
    b();
    printf("这是第4个输出! \n");
}
void b(void) { printf("这是第3个输出! \n"); }
```



```

这是第1个输出!
这是第2个输出!
这是第3个输出!
这是第4个输出!
这是第5个输出!

```

(2) 函数的递归调用

```
#include<stdio.h> //斐波那契数列
```

```
int Fib(int n)
```

```
{
```

```
    if(n==1||n==2) return 1;;
```

```
    return Fib(n-1)+Fib(n-2);
```

```
}
```

```
void main( )
```

```
{
```

```
    int n; long m;
```

```
    printf("请输入要求的是第几项: ");
```

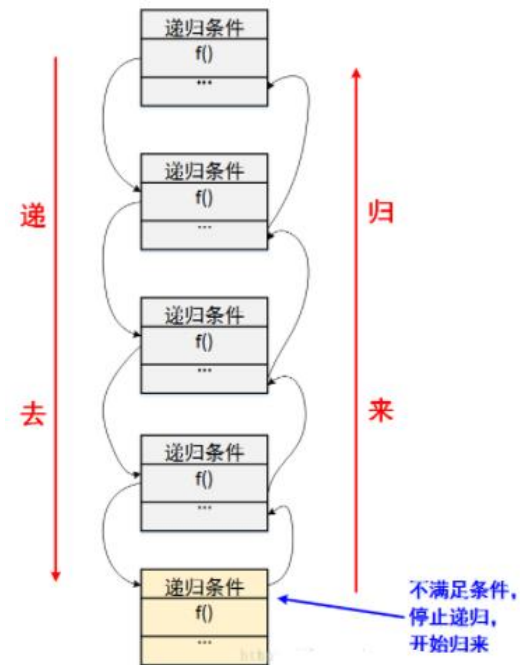
```
    scanf("%d", &n);
```

```
    m=Fib(n);
```

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

```
}
```

```
请输入要求的是第几项: 20
6765
```



//从(0,0)到(m,n)走只能向上或者向右走，一次只能走一步有多少种走法

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

```
int fun(int m, int n)
```

```
{
```

```
    if(m==0&& n==0) return 0;
```

```
    else if(m==0||n==0) return 1;
```

```
        else return (fun(m-1, n)+fun(m, n-1));
```

```
}
```

```
void main( )
```

```
{
```

```
    int m,n; long R;
```

```
    printf("输入两个正整数: ");
```

```
    scanf("%d%d", &m, &n);
```

```
    R=fun(m, n);
```

```
    printf("走法总数=%d.\n", R);
```

```
}
```

```
输入两个正整数: 10 10
走法总数=184756.
```

- 函数调用时所涉及的参数：

- (1) 形式参数（形参）

函数定义时函数名后面的参数，是一个变量。如：

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

在函数调用前，形式上存在，系统不对其分配存储单元；发生函数调用时，系统给形参分配存储单元；调用结束后，立即释放该单元。形参是局部变量

- (2) 实际参数（实参）

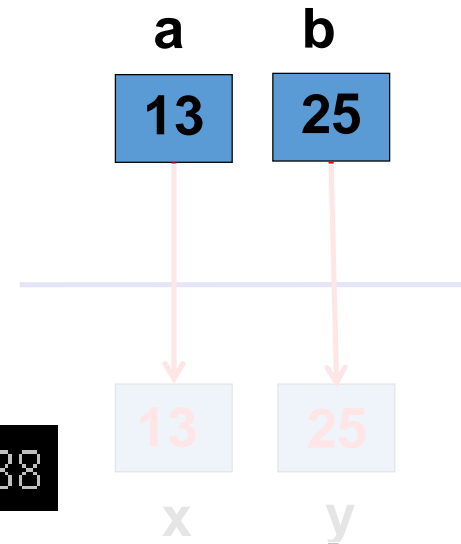
函数调用时函数名后面的参数，一般为具有确定值的常量、变量或表达式。如： sum=Add(23, 3+5);

```
#include<stdio.h>
void main( )
{
    int Add(int, int);
    int a=13, b=25, sum;
    sum=Add(a, b);
    printf("Sum=%d\n", sum);
}
```

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

Sum=38

传值调用



说明:

- (1) 虚实结合的特点为：**单向值传递**
- (2) 实参可以是常量，具有确定值的变量、数组元素或表达式；形参为实参的一个副本，对形参的操作不会影响到实参，即不能通过形参改变实参的值
- (3) 函数调用按照原型进行，实参表达式的值被转换为原型所指定的形参的类型

如： `sum=Add(3.8, 4.6);`

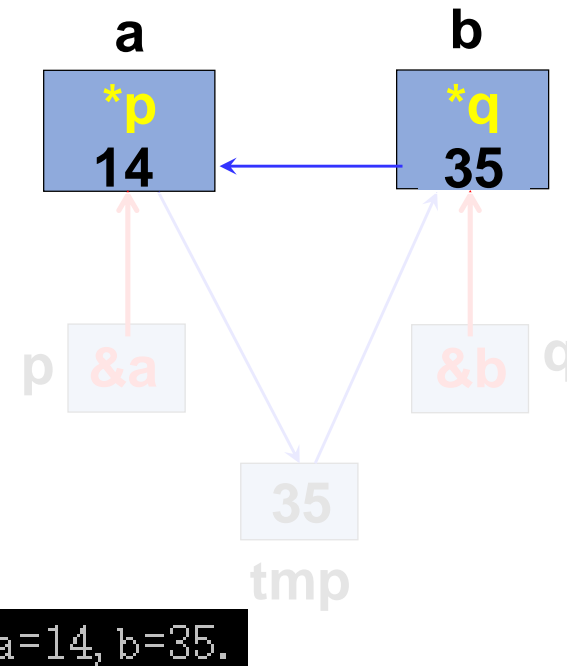
`printf("Sum=%d\n", sum);`

Sum=7


```
#include<stdio.h>
void main( )
{
    void fun1(int *, int *);
    int a=35, b=14;
    fun1(&a, &b);
    printf("a=%d,b=%d.\n", a, b);
}
```

```
void fun1(int *p, int *q)
{
    int tmp;
    tmp=*p; *p=*q; *q=tmp;
}
```

引用调用



说明:

- (1) 虚实结合的特点为：**双向地址传递**
- (2) 变量地址（**指针**）作为实参，形参定义为指针变量，接收传递过来的指针。通过这种方式，在被调函数中就能引用主调函数中定义的变量，达到了函数间**共享内存空间**的功能
- (3) 克服了通过return语句只能带回一个值的弊端，一次可以带回多个值
- (4) 可以用数组名或字符串作为实参

```

#include<stdio.h>
void fun(int x[10]) //形参数组的长度可以不指定，如可改为：int x[]
{
    int i;
    for(i=0;i<10;i++)
        x[i]=x[i]+1;    //形参数组和实参数组共享同一段内存空间
}

```

Shanghai ECUST

```

void main( )
{
    int i, a[10]={0,1,2,3,4,5,6,7,8,9};
    fun(a);
    for(i=0;i<10;i++)
        printf("%3d", a[i]);
    printf("\n");
}

```

1 2 3 4 5 6 7 8 9 10

```
#include<stdio.h>
#include<string.h>
void fun(char x[],char y[])
{
    strcat(x, y);
}
```

```
void main( )
{
    char a[20]="Shanghai";
    fun(a," ECUST");
    puts(a);
}
```

Shanghai ECUST

● 函数的返回值：

- ✓ 通过**return语句**返回一个值给主调函数
- ✓ 格式：**return 表达式**；或 **return (表达式)**；
- ✓ 表达式的数据类型应与函数定义时函数名前面的类型一致；如果不一致，**以函数的类型为准**，即返回值的类型由函数类型决定
- ✓ 当函数中有多个return语句时，执行到哪一个return语句，哪一个起作用

6.4 变量的作用域与生存期

- 作用域和生存期是程序设计中常用的有联系但又不同的两个概念
- 作用域(scope): 变量起作用的范围
- 生存期(lifetime): 变量在内存中存在的时间

• 作用域

- ✓ 是一个**静态的**概念，即在程序的哪些部分可以合法引用变量，表明了变量活动的**空间**
- ✓ 一个变量的作用域由变量的位置决定
- ✓ 从作用域角度，将变量分为：
 - (1) **全局变量**：在函数之外定义的变量
 - (2) **局部变量**：在函数内或复合语句中定义的变量

例：用辗转相除法求两个数的最大公约数和最小公倍数

算法分析：

- ✓ 辗转相除法,又叫欧几里得算法 (Euclidean algorithm)，目的是求两个正整数的最大公约数。它是已知最古老的算法，可追溯至公元前300年前
- ✓ 算法基于下面定理：假设两个正整数 a 和 b ($a > b$)，它们的最大公约数等于 a 除以 b 之余数 r 和 较小数 b 之间的最大公约数；最小公倍数是 $ab/\text{最大公约数}$
- ✓ 如： $a=25$ 和 $b=10$; $r=25\%10$; $r=10\%5$; 最大公约数为**5**


```
#include "stdio.h"
```

```
int gys=1, gbs=0; //全局变量gys、gbs
```

```
void fun1 ( int x, int y ) //局部变量x、y
{ int t, r; //局部变量t、r
  if (y>x) { t=x; x=y; y=t; }
  while((r=x%y) !=0) { x=y; y=r; }
  gys=y;
}
```

x、y、r、t仅在此函数内有效

gys、gbs
在此范围有效

```
int fun2 ( int x, int y ) //局部变量x、y
{ int gbs; //局部变量gbs
  gbs=x*y/gys;
  return gbs;
}
```

x、y、gbs仅在此函数内有效

```
void main( )
```

```
{ int a, b; //局部变量a、b
```

```
scanf("%d%d", &a, &b);
```

```
fun1(a, b);
```

```
gbs=fun2(a, b);
```

```
printf("最大公约数和最小公倍数分别是： %d, %d\n", gys, gbs);
```

```
}
```

25 10

最大公约数和最小公倍数分别是： 5, 50

a、b仅在此函数内有效

说明:

- (1) 函数之外定义的变量为全局变量，其作用域为定义点开始到文件结束；编译系统在编译阶段为全局变量分配内存空间，程序执行结束时释放空间
- (2) 在函数内或复合语句中定义的变量为局部变量，其作用域为所在的函数或复合语句内；程序运行过程中，编译系统为局部变量分配内存空间，函数调用结束时释放空间
- (3) 全局变量和局部变量可以同名，在局部变量的作用范围内，全局变量不起作用

- (4) 对于全局变量，可以通过加extern的原型声明来在**本文件内扩展**其作用域，提前获得全局变量的使用权：**extern 数据类型 全局变量名列表；**
- (5) 对于由多个源文件组成的程序，可以通过加extern的原型声明将一个全局变量的作用域**扩展到其他文件**：在一个文件中定义，在其他文件中声明
- (6) 可通过static将全局变量的作用域限制在本文件中
- (7) 全局变量的使用，减少了通过函数参数传递数据带来的系统开销，但破坏了函数的独立性，降低了函数的可移植性，使代码的可读性降低，不利于程序调试。因此，**不提倡大量使用全局变量**

```

#include "stdio.h"
void fun1 ( int x, int y ) //局部变量x、y
{ int t, r;    //局部变量t、r
  extern int gys;    //声明全局变量gys, 在本文件中扩展其作用域
  if (y>x) { t=x; x=y; y=t; }
  while((r=x%y) !=0) { x=y; y=r; }
  gys=y;
}
int gys, gbs; //定义全局变量gys、gbs
int fun2 ( int x, int y ) //局部变量x、y
{ int gbs; //局部变量gbs
  gbs=x*y/gys;
  return gbs;
}

void main( )
{ int a, b; //局部变量a、b
  scanf("%d%d", &a, &b);
  fun1(a, b);
  gbs=fun2(a, b);
  printf("最大公约数和最小公倍数分别是: %d, %d\n", gys, gbs);
}

```

程序变形一

```
// my_f1.c
void fun1 ( int x, int y ) //局部变量x、 y
{ int t, r;    //局部变量t、 r
  extern int gys;    //声明全局变量gys
  if (y>x) { t=x; x=y; y=t; }
  while((r=x%y) !=0) { x=y; y=r; }
  gys=y;
}
```

```
// my_f2.c
int fun2 ( int x, int y ) //局部变量x、 y
{ extern int gys;
  int gbs;    //局部变量gbs
  gbs=x*y/gys;
  return gbs;
}
```

```
// my_f3.c
#include "stdio.h"
int gys, gbs;
void main()
{ void fun1(int x, int y); int fun2(int x, int y);
  int a, b;    //局部变量a、 b
  scanf("%d%d", &a, &b);
  fun1(a, b);
  gbs=fun2(a, b);
  printf("最大公约数和最小公倍数分别是: %d, %d\n", gys, gbs);
}
```

程序变形二：工程文件

工程文件方式

- 建立工程

文件->新建（选择“工程”选项卡，选中Win32 Console Application选项，并在“工程”文本框输入工程名字->确定->完成->确定）

- 添加源程序

“工程”->“添加工程”->Files

- 对各源程序单独编译，通过连接合并起来，然后执行

只能用于f1文件

```
// f1.c
static int A;
int main ( )
{
    .....
}
```

f2文件将不能使用

```
// f2.c
extern int A;
void fun (int n)
{
    .....
    A=A*n;
    .....
}
```

```
// my_1.c
#include "stdio.h"
void fun1 ( int x, int y ) //局部变量x、y
{ int t, r; //局部变量t、r
  extern int gys; //声明全局变量gys, 在本文件中扩展其作用域
  if (y>x) { t=x; x=y; y=t; }
  while((r=x%y) !=0) { x=y; y=r; }
  gys=y;
}
int gys, gbs; //定义全局变量gys、gbs
int fun2 ( int x, int y ) //局部变量x、y
{ int gbs; //局部变量gbs
  gbs=x*y/gys;
  return gbs;
}
```

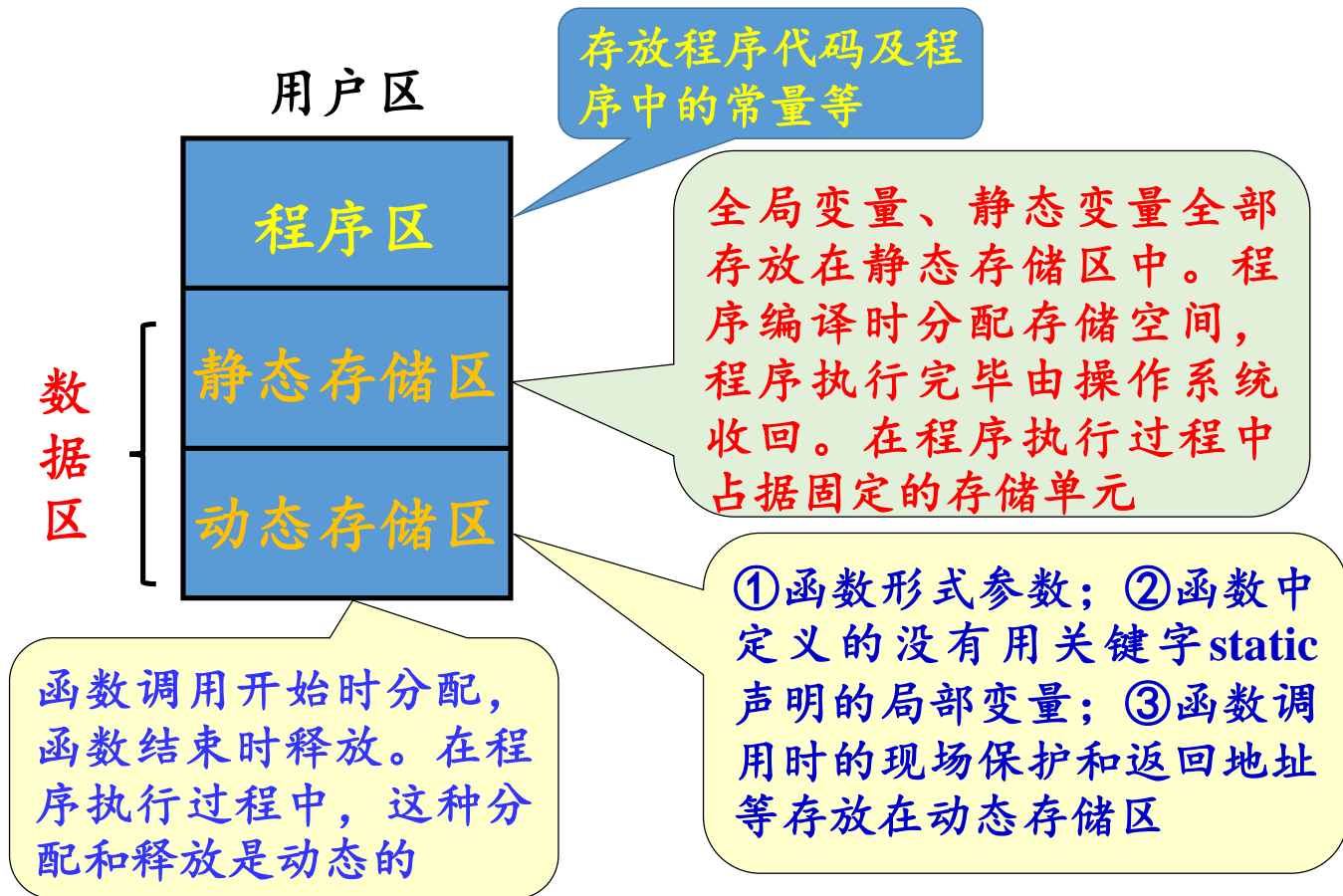
程序变形三：文件包含

```
// my.c
#include "stdio.h "
#include "F:\\CAI\\计算机程序设计\\2018\\样例程序\\my_1.c"
void main( )
{ int a, b; //局部变量a、b
  scanf("%d%d", &a, &b);
  fun1(a, b);
  gbs=fun2(a, b);
  printf("最大公约数和最小公倍数分别是: %d, %d\\n", gys, gbs);
}
```


● 生存期

- ✓ 是一个**动态的**概念，表明了变量存在的时间
- ✓ 一个**变量的生存期由变量的存储类别**决定。

存储类别的作用是规定系统应该为该变量分配什么类型的存储空间，存储空间类型决定了变量什么时候在内存中创建，什么时候释放该空间



✓ **存储类别**：包括自动、寄存器、静态、外部四种，分为临时的和永久的两类：

(1) 临时的：自动、寄存器

自动(auto)：用于指定局部变量的存储类型，
可以省略，动态变量，值为随机

格式：**[auto] 类型名 变量名;**

例：`auto int x, y=1, z=2;`

寄存器(register): 用于指定局部变量的存储类型, 请求编译器尽量直接分配CPU中的寄存器, 满则分配内存。速度快, 主要用于循环控制变量, 但寄存器数量有限

例: ...
auto sum=0;
register int i;
for(i=0; i<100; i++) sum+=i;
...

(2) 永久的：静态、外部

外部(extern)：函数之外定义的变量为外部变量或全局变量，外部变量的默认存储类型是extern。

extern既可以用来扩展全局变量的作用域，也可以用来定义外部变量的存储类型（定义外部变量时，**extern**通常省略）

静态(static): 用于指定全局变量或局部变量，但意义不同，静态变量

格式: **static 类型名 变量名;**

静态全局变量: 静态存储，初值为0，该变量只能在本文件使用，其他文件无权使用

静态局部变量: 静态存储，编译时分配空间，初始化只进行一次，调用结束不释放空间，程序执行完毕时，其生存期才结束

例：输入一个正整数 n ，计算 $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}$
，要求 n 从键盘上输入，结果保留4位小数。

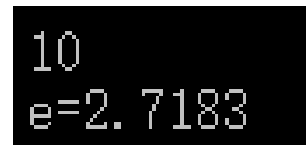
输入输出示例：

Input n:10↵

e=2.7183

```
#include "stdio.h "
float fun(int k)
{  static float f=1;
   f=f*k;
   return f;
}

void main( )
{  int i, n;
   float e=1.0;
   scanf("%d", &n);
   for(i=1; i<=n; i++) e=e+1/fun(i);
   printf("e=%.4f\n", e);
}
```



```
10
e=2.7183
```


• 外部函数和内部函数

对于由多个源文件组成的程序

- ✓ 如果一个函数可以被其他源文件调用，该函数就称为外部函数

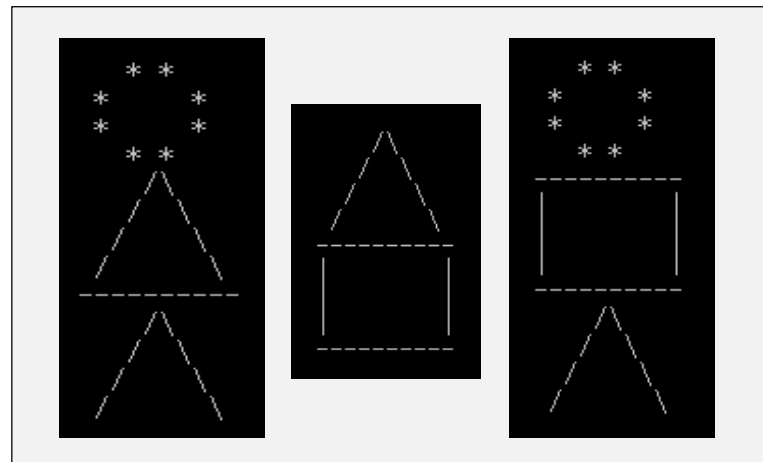
[extern] 函数类型 函数名(形式参数)

- ✓ 如果一个函数只能被本文件中的其他函数调用，该函数就称为内部函数

static 函数类型 函数名(形式参数)

6.5 模块化程序设计举例

例：绘制如下图形，用工程文件及文件包含两种方法实现



女孩

房子

男孩

画基本组件的函数：圆

```
/* Draws a circle */  
void draw_circle(void)  
{  
    printf("  * * \n");  
    printf(" *   *\n");  
    printf(" *   *\n");  
    printf("  * * \n");  
}
```

画基本组件的函数：交叉线

```
/*Draws intersecting lines*/  
void draw_intersect(void)  
{  
    printf("    /\ \n");  
    printf("  /  \ \n");  
    printf(" /   \ \n");  
    printf("/    \\ \n");  
}
```

画基本组件的函数：横线

```
/* Draws a base line */  
void draw_base(void)  
{  
    printf(" -----\\n");  
}
```

画基本组件的函数：平行线

```
/* Draws a parallel lines */  
void draw_parallel(void)  
{  
    printf(" |      |\n");  
    printf(" |      |\n");  
    printf(" |      |\n");  
}
```

画组合图形的函数：女孩

```
/*Draws girl*/  
void draw_girl(void)  
{  
    draw_circle( );      // Draw a circle  
    draw_triangle( );    // Draw a triangle  
    draw_intersect( );   // Draw intersecting lines  
}
```

画组合图形的函数：房子

```
/*Draws house*/  
void draw_house(void)  
{  
    draw_triangle( ); // Draw a triangle  
    draw_parallel( ); // Draws a parallel lines  
    draw_base( );     // Draw a base  
}
```


画组合图形的函数： 男孩

```
void draw_boy(void)
{
    draw_circle( );    // Draw a circle
    draw_base( );      // Draw a base
    draw_parallel( );  // Draws a parallel
    draw_base( );      // Draw a base
    draw_intersect( ); // Draw intersecting lines
}
```

```
/* This is a program to output a picture of a family */  
#include <stdio.h>  
void draw_circle(void);    // Draws a circle  
void draw_intersect(void); // Draws intersecting lines  
void draw_base(void);     // Draws a base line  
void draw_triangle(void); // Draws a triangle  
void draw_parallel(void); // Draws a parallel lines  
void draw_girl(void);     // Draws a girl  
void draw_house(void);    // Draws a house  
void draw_boy(void);      // Draws a boy  
  
void main(void)  
{ draw_girl( ); draw_house( ); draw_boy( ); }
```

Homework

- 实践

实验4 模块化程序设计

- 作业

《教材》： P218-219 5、12、15