第 2 讲

程序设计基础

主要内容

- 计算机程序的基本结构
- 结构化程序的构成要素
- 计算机程序设计基本方法与技术

2.1 计算机程序的基本结构

1. 函数的结构
2. 源程序文件结构
3. 程序结构

1. 函数的结构

函数首部

{

  定义及声明部分

  执行部分　　//由基本语句或控制语句组成

}

结构化程序设计

- 三种基本结构：即用顺序、选择、循环这三种基本结构组成程序。1966年由Bobra和Jacopini提出
- 结构化程序便于编写、阅读和维护

三种基本结构的共同特点：

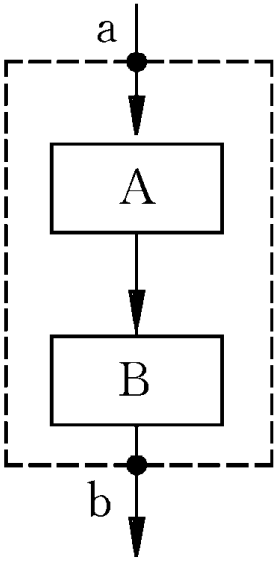
(1) 只有一个入口、一个出口

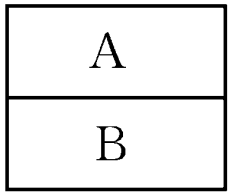
(2) 结构内的每一部分都有机会被执行到

(3) 结构内不存在”死循环”(无终止的循环)

- **顺序结构：** 由语句序列组成，程序执行时，按照从上而下顺序．一条一条地顺序执行

流程图

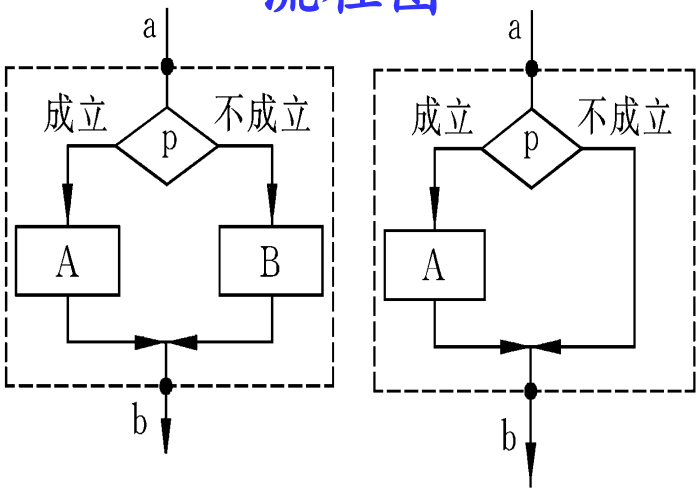




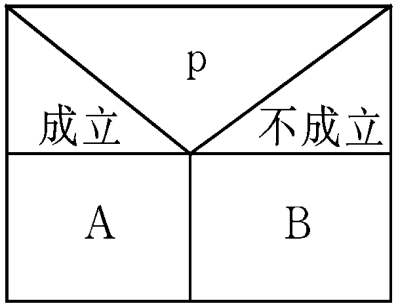
N-S图

- **选择结构：** 根据一定的条件决定执行哪一部分的语句序列

流程图

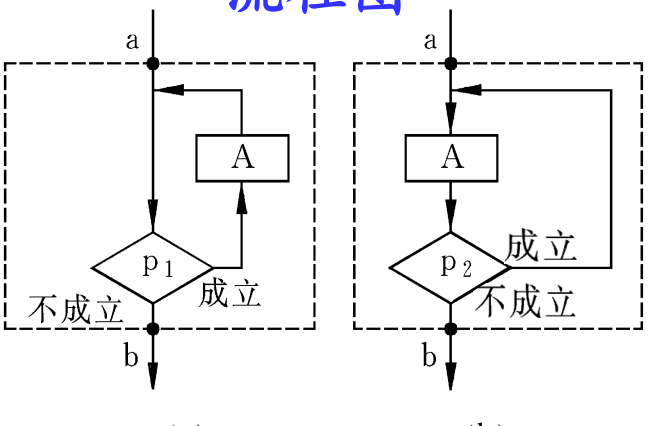


N-S图



- **循环结构：**使同一个语句组根据一定的条件执行若干次

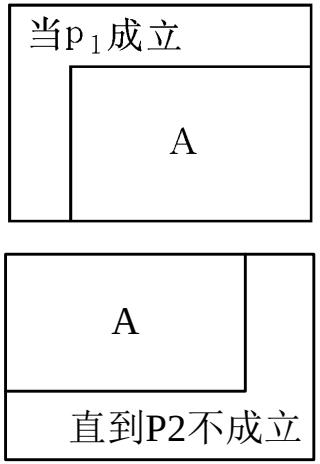
流程图



(a)　　(b)

当型(while型)循环　**直到型(until型)循环**

N-S图



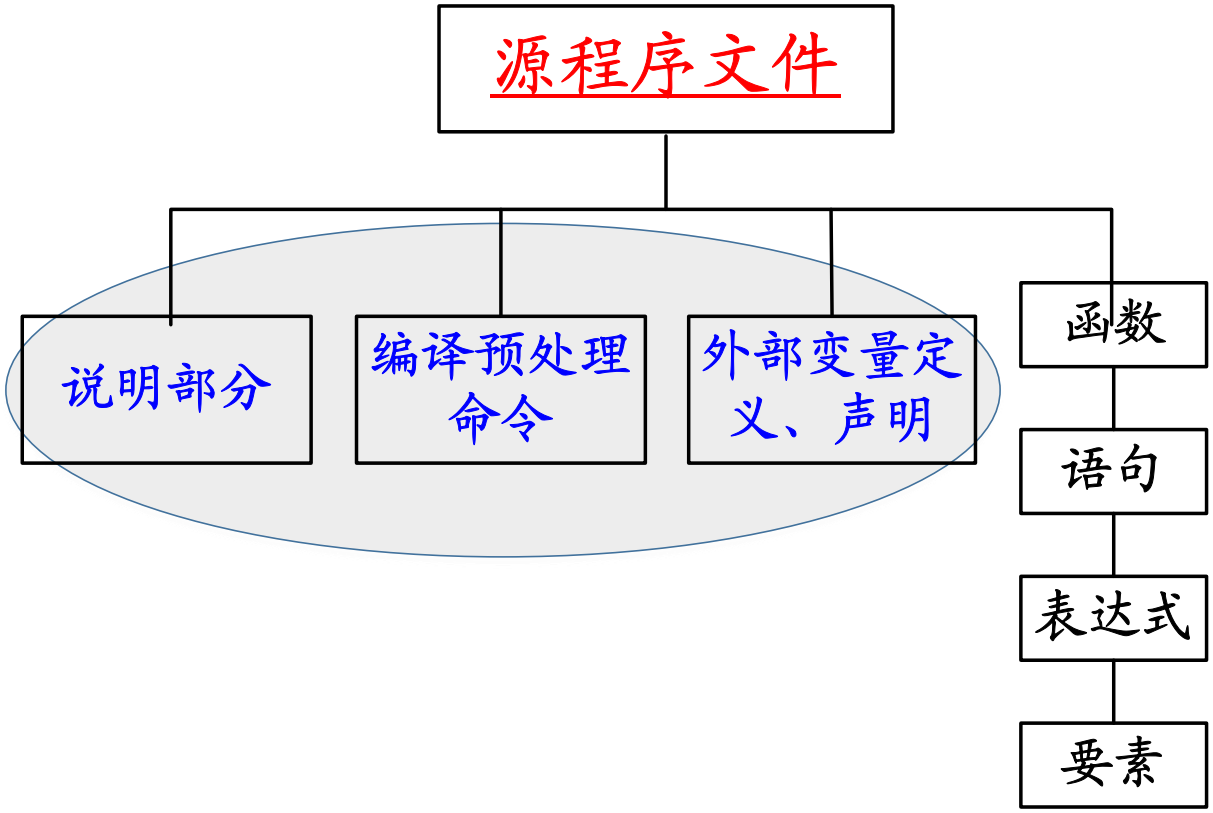
小结：

- 由三种基本结构组成的程序结构，可以解决任何复杂的问题
- 由基本结构所构成的程序属于“结构化”程序，它不存在无规律的转向，只在本结构内才允许存在分支和向前或向后的跳转

2. 源程序文件结构

源程序文件一般包括：

- 程序说明部分
- 编译预处理命令
- 外部变量定义、声明等
- 函数定义


源程序文件
说明部分
编译预处理命令
外部变量定义、声明
函数
语句
表达式
要素

编译预处理

- 以#开头的命令，C编译系统对程序进行正常的编译之前，先对这些特殊的命令进行预处理，然后将预处理的结果和源程序一起再进行通常的编译处理，以得到目标代码
- C提供的预处理功能：宏、文件包含、条件编译
- 编译预处理命令一般位于程序的开头，每行只能写一条预处理命令

- 宏定义
  - 不带参数的宏定义: **#define 宏名 宏体**

    #define PI 3.14
  - 带参数的宏定义: **#define 宏名(参数表) 宏体**

    #define MAX(x,y) x>y?x:y
  - 终止宏定义的作用域: **#undef(宏名)**

    #undef(PI)

- 文件包含
  - 是指一个源文件可以将另一个源文件的全部内容包含进来
  - 两种文件包含形式: **#include<文件名>**
    #include “文件名”
  - 把指定的文件内容嵌入到另一个文件中
  - 通常把一些常数和宏定义编写成.h文件，并用#include命令放置到任何需要的源文件中

- 条件编译
  - 对源程序中某一部分内容只在满足一定条件时才进行编译

```
#if  常量表达式
        程序段1
#else
        程序段2
#endif
```

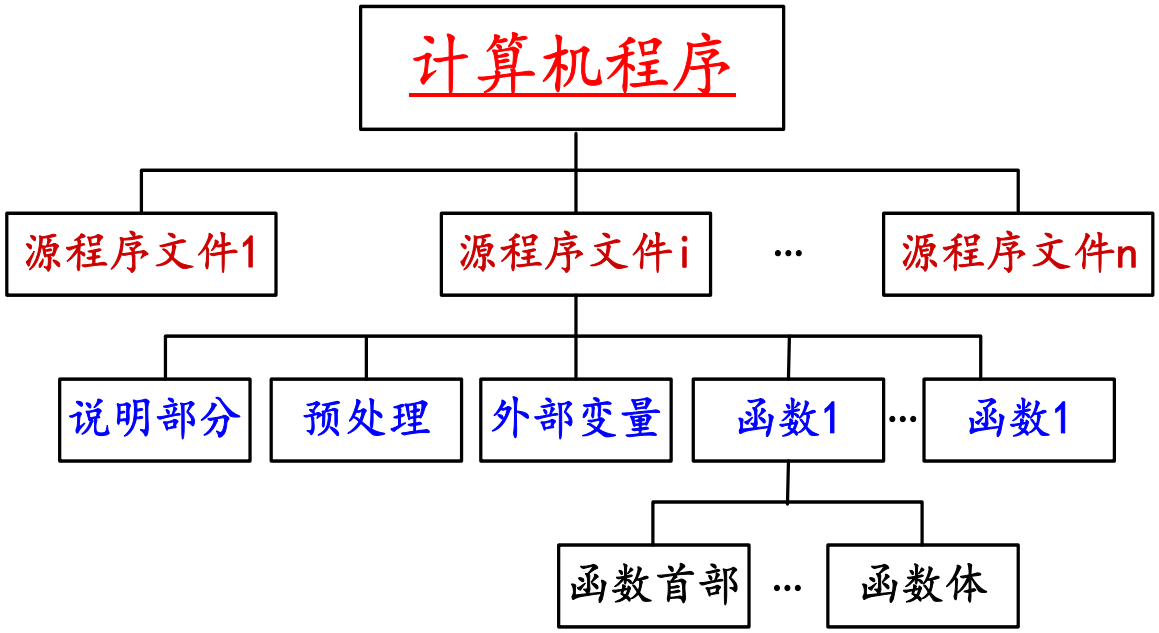
```
#if  常量表达式
        程序段
#endif
```

```
#if  常量表达式1
        程序段1
#elif  常量表达式2
        程序段2
#else
        程序段3
#endif
```

```
#ifdef  标识符
        程序段1
#else
        程序段2
#endif
```

```
#ifndef  标识符
        程序段
#endif
```

3. 计算机程序结构



程序设计风格

- 结构化编码强调程序风格和程序结构的规范化，提倡按照现代软件工程的规范进行编码
- 清晰的程序结构和风格，易读、易维护，重用性、可靠性和效率高

- 程序风格
  - 代码行：每行写一条语句，每个常量定义占一行
  - 对齐与缩进：同一层次代码对齐，用缩进体现代码的层次性
  - 空行与空格：在函数、功能模块之前适当加空行，在代码行中添加一定的空格
  - 长行拆分：过长的代码行可适当拆分成几行
  - 注释：所有程序、所有函数都从注释开始，对程序段、复合语句等添加注释

例：中国古代数学“鸡兔同笼”问题

“今有鸡兔同笼，上有三十五头，下有九十四足，问鸡兔各几何”

1. 分析问题

分析问题需要明确输入，也就是要处理的数据；明确输出，也就是结果；解决方案的任何附加要求或约束条件

问题输入：总脚数94，总头数35

问题输出：鸡数a和兔数b

关系公式：$a + b = 35$

$2 \times a + 4 \times b = 94$

推导公式：

$b = (94 - 2 \times 35) \div 2$

$a = 35 - b$

2. 设计算法

鸡兔同笼问题

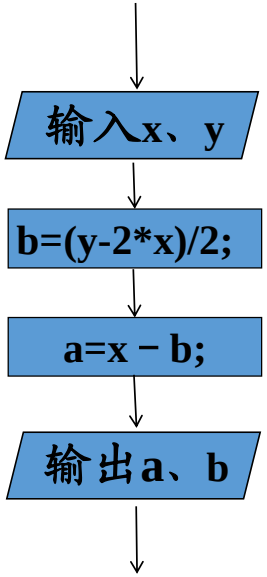
假设:

x代表总头数，

y代表总脚数，

a鸡数，

b兔数



3.编写代码

```
/* 文件名：鸡兔.c 主要功能：计算鸡兔数 */
#include<stdio.h>      //包含头文件
int x, y, a, b;        //定义4个外部变量
void main( ){          //主函数
      Input_jitu();
      b=(y-2*x)/2;     //计算兔子数
      a=x-b;           //计算鸡数
      Output_jitu();   //输出结果
 }
```

```
#include<stdio.h>      // 鸡兔input.c
extern int x, y;

void Input_jitu( )
{
   printf("Head (x) and foot numbers (y):");
   scanf("%d%d", &x, &y);  //输入数据
}
```

```
#include<stdio.h>    // 鸡兔Output.c
extern int a, b;

void Output_jitu()
{
   printf("鸡:%3d 兔:%3d\n", a, b);
 //输出结果
}
```

4. 运行求解

```
Head number (x) and foot numbers (y):35 94
鸡: 23  兔: 12
```

2.2 结构化程序的构成要素

（1）常量、变量

（2）标识符

（3）运算符

（4）表达式

（5）语句

（6）注释

（1）常量、变量

常量

在程序运行过程中值不能被改变的量。例如：

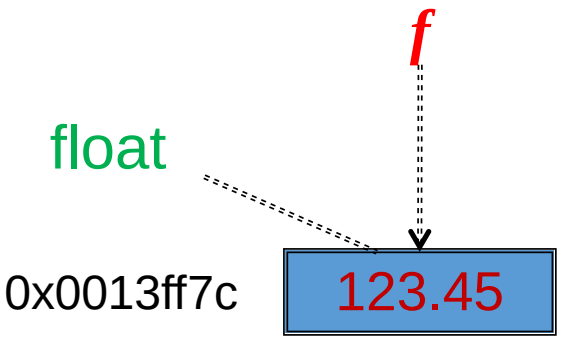
- 整型常量：如12345，023，0x12，-345，3L
- 实型常量
  - ◆十进制小数形式：如 0.34, -56.79, 0.0, 3.1415f
  - ◆指数形式：如 12.34e3 (代表12.34×10^3)
- 字符常量：如 ‘?’
  - 转义字符：如 ‘\n’, ‘\r’ ’, ‘\a’
- 字符串常量：如 “Shanghai”
- 符号常量：#define PI 3.14159265

变量

用以存储数据，且在程序运行期间值可以改变的量

变量的属性

- 名称
- 数据类型
- 存储位置
- 值



变量的定义

变量必须先定义，后使用；定义变量时指定该变量的名字（符合标识符的命名规则）和类型

数据类型　变量表

```
float f;
//以华氏度表示的温度
 float c;
//以摄氏度表示的温度
```

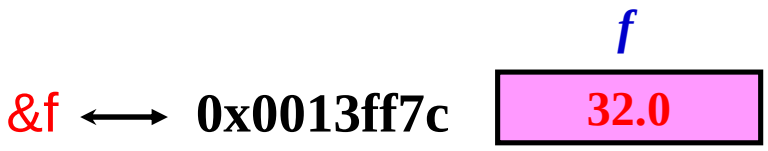
```
float f, c;
```

- 定义通常放在函数起始处，在任何可执行语句之前
- 变量的初值—随机值
- 变量的初始化

```
float  f=32;
```

变量的操作

- 将数据存入变量： f=32;
- 取得变量中保存的值：c=5.0/9*(f-32);
- 求变量的地址(指针)：&f
  - 变量名实际上是以一个名字代表的一个存储地址
  - 从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据



（2）标识符

命名规则

- 字母（a~z、A~Z）、数字（0~9）、下划线（_）组成，且必须由字母或下划线开头
- 不能把C语言的关键字（如if、for、while）作为标识符
- 对大小写敏感
- 应“见名知意”

（3）运算符

C运算符（34个）

1) 算术运算符　　（+ - * / % ++ --）
2) 赋值运算符　　（=及其扩展赋值运算符）
3) 强制类型转换运算符　　（(类型)）
4) 关系运算符　（> < == >= <= !=）
5) 逻辑运算符　　（! && ||）
6) 位运算符　　（<< >> ~ | ^ &）

7) 条件运算符 （？：）如：a=3<5?3:5

8) 逗号运算符 （，）

9) 指针运算符 （*和&）

10) 求字节数运算符 （sizeof）

11) 成员运算符 （. ->）

12) 下标运算符 （[ ]）

13) 其他 （如函数调用运算符( )）

运算符的优先级和结合性

- 混合运算中谁先谁后由优先级确定
- 同等优先级下，从左向右计算还是从右向左计算由结合性确定

不同类型数据间的混合运算：

1) +、-、*、/ 运算的两个数中有一个数为float或double型，结果是double型。系统将float型数据都先转换为double型，然后进行运算
2) 如果int型与float或double型数据进行运算，先把int型和float型数据转换为double型，然后进行运算，结果是double型
3) 字符型数据与整型数据进行运算，就是把字符的ASCII代码与整型数据进行运算

（4）表达式

用运算符和括号将运算对象（也称操作数）连接起来的、符合C语法规则的式子，称为C的表达式

1) 算术表达式
2) 赋值表达式
3) 关系表达式
4) 逻辑表达式
5) 位运算表达式
6) 逗号表达式
7) 条件表达式

（5）语句

程序的主体部分是由语句组成的，程序的功能也是由执行语句实现的；语句也是函数的组成单位。语句的标志是分号

1) 控制语句： if、switch、for、while、do…while、continue、break、return、goto等
2) 函数调用语句
3) 表达式语句
4) 空语句
5) 复合语句

（6）注释

注释是程序设计中一项非常重要的内容。

程序中常用的注释内容

1) 程序的名称
2) 程序的功能
3) 程序的设计思路与特点
4) 编程人及其合作者
5) 程序的编写时间、版本
6) 其他需要说明的信息

C语言中注释的方法：

（1）块注释

/* 注释内容，可以跨行书写 */

（2）行注释

// 注释内容，不能跨行书写

2.3 计算机程序设计基本方法与技术

结构化程序设计方法

- 基于“分而治之”思想
- 以模块化设计为中心
- 采用自顶向下、逐步求精的方法
- 通过结构化编码和限制goto语句

设计出逻辑清晰、结构合理、性能优良的计算机程序

分而治之

- Divide and conquer
- 处理策略：

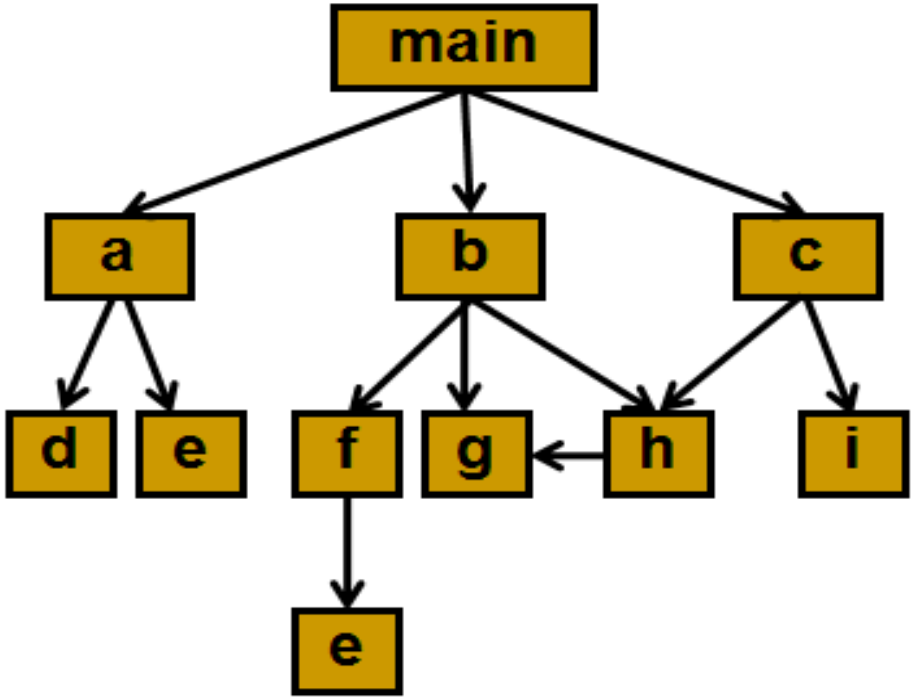
  I. 把一个复杂的大问题分解为两个或多个规模较小的子问题

  II. 分别解决每个子问题

  III.把各子问题的解组成整个问题的解

模块化设计

- Modularity design
- 采用“组装”的办法简化程序设计过程
- 模块的实现——函数，即事先编好一批实现各种不同功能的函数
- 每一个函数用来实现一个特定的功能
- 把它们保存在函数库中，需要时直接用，减少重复编写代码的工作量


main
a
b
c
d
e
f
g
h
i
e

自顶向下

- Top-Down Design，与自底向上方法对应
- 设计程序时，先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标
- 先从最上层总目标开始设计，逐步使要解决的问题详细化。即步步深入、逐层扩展，直到所有步骤可以详细到能够转换为程序语句
- 一般利用流程图或伪代码完成细化过程

逐步求精

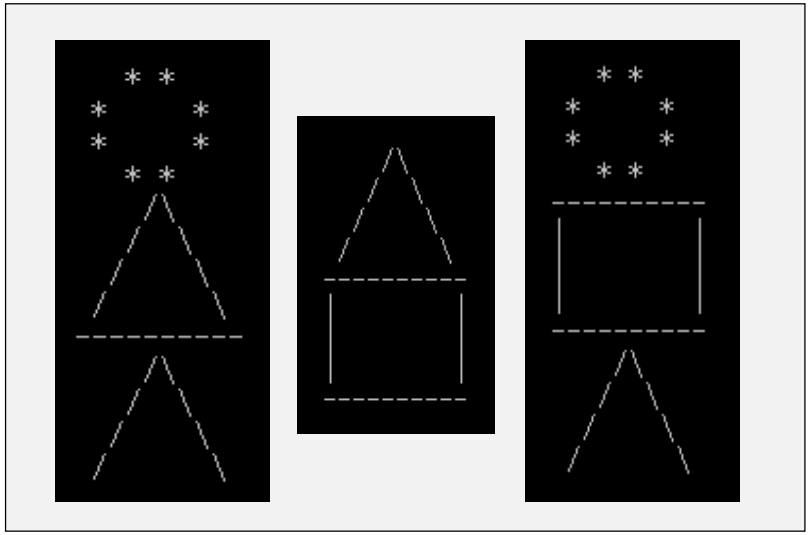
- Stepwise refinement
- 即把大任务分割成小的更容易控制的块，再继续细分为更小的任务，直到所有的小任务能很容易实现
- 设计时，首先考虑程序的整体结构而忽视细节，然后逐步地、一层一层地细化程序，直到细节可直接利用程序设计语言描述

- 在利用C语言设计较大程序时，往往把它分为若干个程序模块，每一个模块包含一个或多个函数，每个函数实现一个特定的功能
- 利用函数可减少重复编写程序段的工作量，方便地实现模块化设计。可以使用库函数，也可自己定义函数
- 主函数调用其他函数，其他函数可互相调用。一个函数可被一个或多个函数调用任意多次

划分子模块的原则

- 子模块一般不超过50行
- 划分子模块时要注意模块的独立性，使一个模块完成一项功能，耦合性愈少愈好

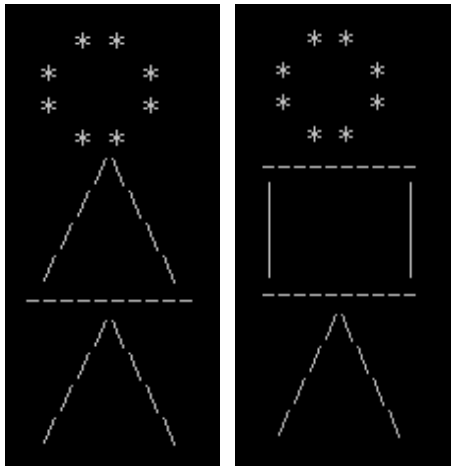
例：绘制如下图形



女孩　　房子　　男孩

分析：

- 基本结构组件
  - ✓ 圆
  - ✓ 交叉线
  - ✓ 横线
  - ✓ 平行线



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");
}
```

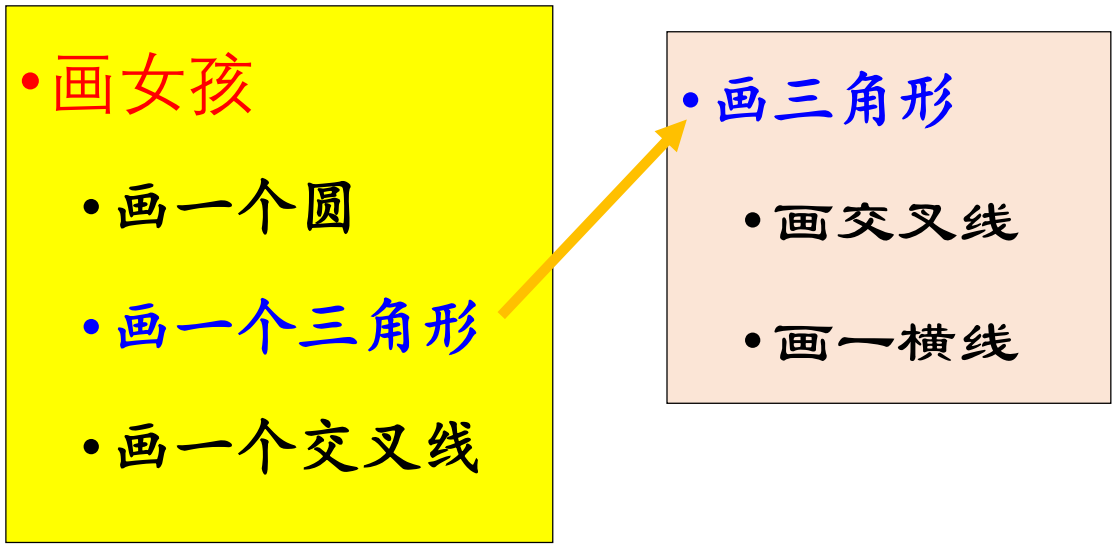
画基本组件的函数：横线

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

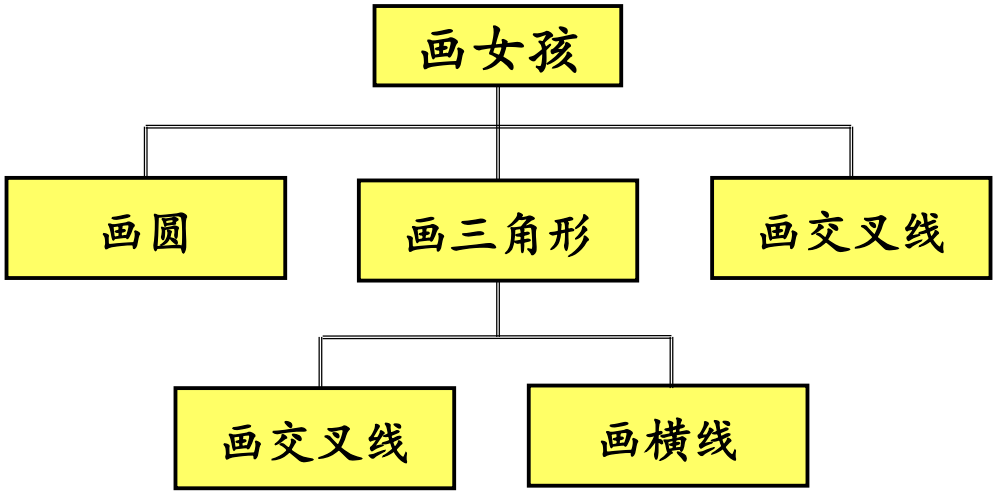
画基本组件的函数：平行线

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

设计（以绘制女孩为例）：



模块结构图：



画组合图形的函数：三角形

```
/* Draws a triangle */
void  draw_triangle(void)
{
      draw_intersect();
      draw_base();
}
```

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
}
```

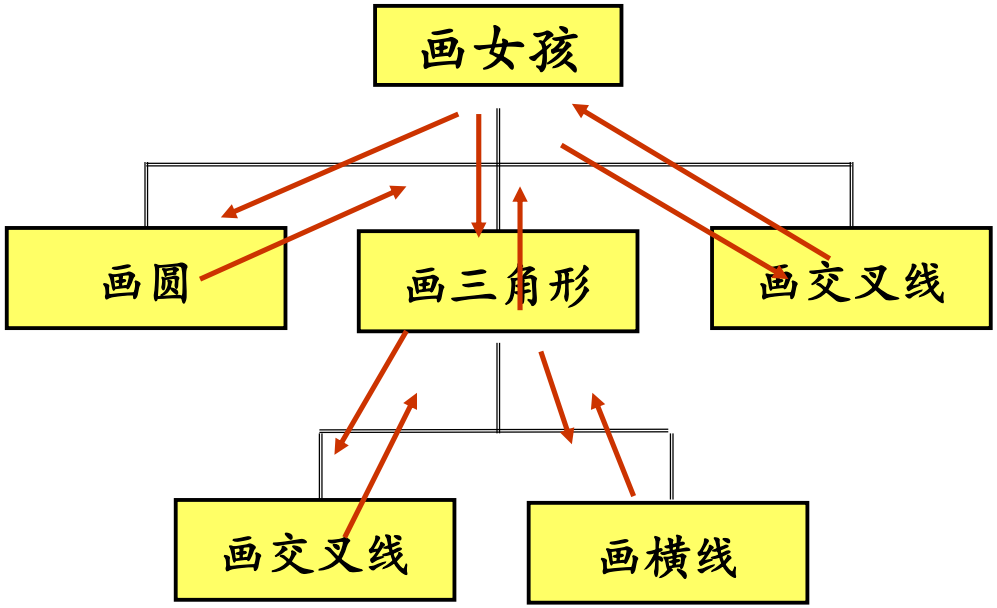
```
/* This is a program to output a picture of a girl */
#include <stdio.h>
void draw_circle(void);      // Draws a circle
void draw_intersect(void); // Draws intersecting lines
void draw_base(void);       // Draws a base line
void draw_triangle(void);   // Draws a triangle
void draw_girl(void);         // Draws a girl

void main(void)
{
   draw_girl( );
}
```

执行流程：

- 程序的执行顺序
  - 从main函数开始执行，执行完main函数结束
  - 遇到调用函数，执行转向被调用函数，执行完子函数，返回调用处，继续向下执行

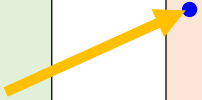
- 程序的书写顺序
  - 函数书写顺序与执行循序无关
  - 函数定义写在函数调用之前

执行示例：



设计（绘制房子）：

- 画房子
  - 画一个三角形
  - 画一个平行线
  - 画一个横线



- 画三角形 √
  - 画交叉线
  - 画一横线

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
}
```

设计（绘制男孩）：

- 画男孩
  - 画一个圆
  - 画一个横线
  - 画一个平行线
  - 画一个横线
  - 画一个交叉线

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();  }
```

结构化程序设计基本技术

- 缩减技术
- 递归技术
- 枚举技术
- 嵌套技术

设计技术	设计机理
缩减技术	大事化小、小事化了
递归技术	以退为进、以简驭繁
枚举技术	选定范围、逐次排查
嵌套技术	内外有别、逐层推进

缩减技术

- 基本思想：大事化小、小事化了
  - 大事化小
    - ✓结构递归：将问题加工成规模压缩了的同类问题，具有清晰的迭代递推结构
    - ✓规模递减：每一步加工后的问题规模相比之前小1
  - 小事化了
    - ✓问题的规模变得足够小可直接转换成相应的语句

- 适用于递推、迭代等过程
  - 顺推
    - ✓累加、累乘、多项式求值、数列求和（斐波那契数列）…
  - 逆推
    - ✓猴子吃桃、上台阶问题
  - 迭代
    - ✓方程求根

$$\sum_{n=1}^{100} n$$

```
…
for(i=1, sum=0; i<=100; i++)
    sum=sum+i;
…
```

$5!$

```
…
for(i=1,sum=1; i<=5; i++)
    sum=sum*i;
…
```

$$\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

```
pi=3.14159065
```

```
#include<stdio.h>
#include<math.h>
void main( ) {
    int sign=1;
    double pi=0, n=1, term=1;
    while(fabs(term)>=1e-6)   {
        pi=pi+term;
        n=n+2;
        sign=-sign;
        term=sign/n;
     }
    printf("pi=%10.8f\n", pi*4);
}
```

例： 猴子吃桃问题

“一只猴子摘了若干桃子，每天吃现有桃子的一半多一个，到第10天只有一个桃子，问桃子有多少个?

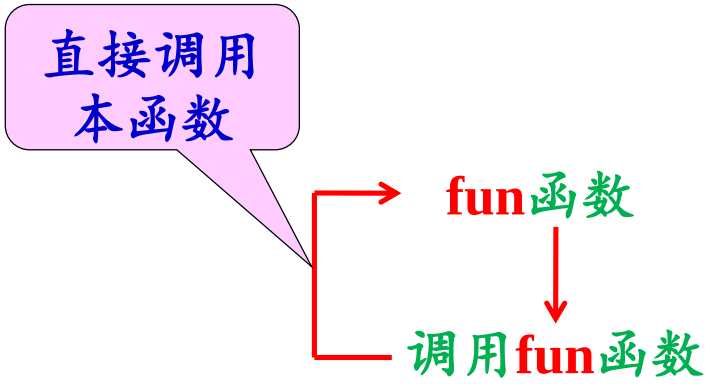
```
#include<stdio.h>
int main( ) {
    int i, n=1;
    for(i=9; i>=1; i--)
       n=(n+1)*2;
    printf(“ Peach=%d\n", n);
    return 0;
}
```

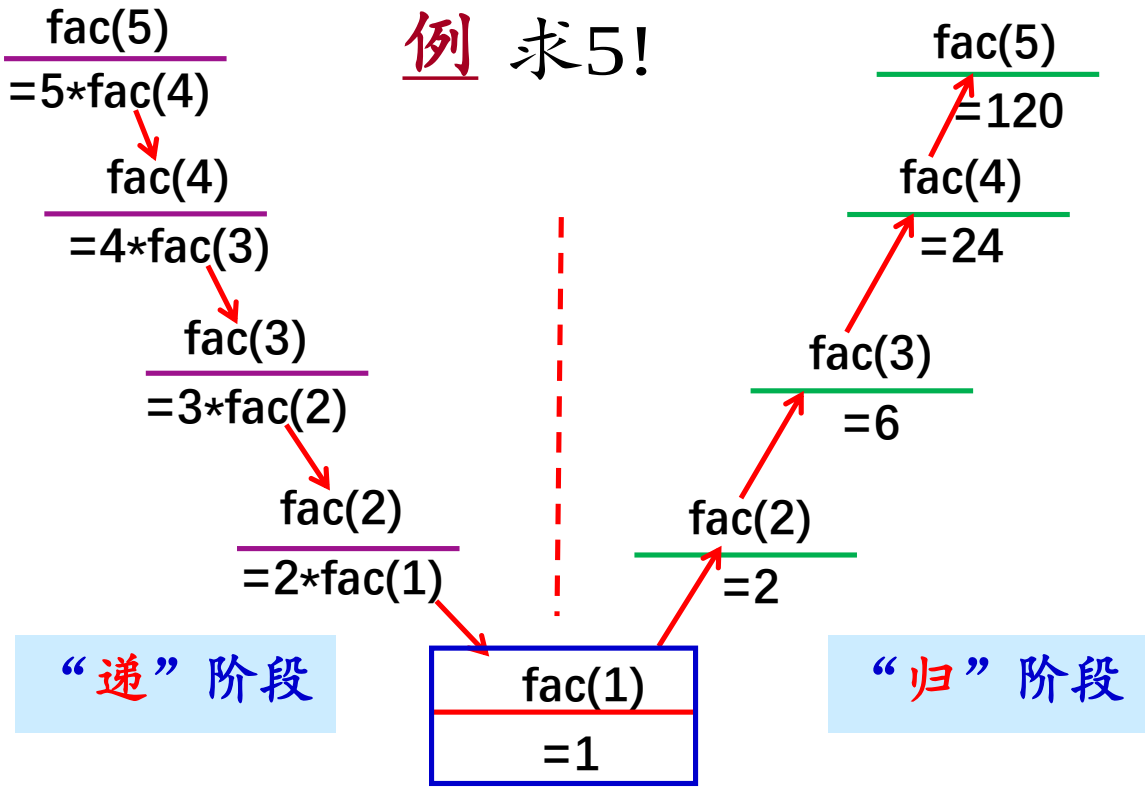
```
 Peach=1534

Process returned 0 (0x0)   execution time : 0.035 s
```

递归技术

- 函数的递归调用：Recursion，在调用函数的过程中又出现直接或间接地调用该函数本身

```
int fun(int x)
{
    int y,z;
    z=fun(y);
    return (2*z);
}
```



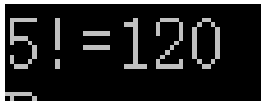

例 求5!
fac(5)
=5*fac(4)
fac(4)
=4*fac(3)
fac(3)
=3*fac(2)
fac(2)
=2*fac(1)
fac(1)
=1
fac(2)
=2
fac(3)
=6
fac(4)
=24
fac(5)
=120
“递”阶段
“归”阶段

- 基本思想：以退为进、以简驭繁
  - 以退为进
    - ✓向下传递：直接求解有困难，故先逐次向下传递，一直传递到“最简单”的操作（能方便完成的操作）
    - ✓向上回归：利用已知条件这个返回替代未知
    - ✓一递一归，有去有回，正反融合，你中有我、我中有你
  - 以简驭繁
    - ✓以简单的递归公式替代原先复杂而抽象的计算
    - ✓把问题转换为规模缩小了的同类问题的子问题

- 性能特点
  - 递归技术是分治策略的最好应用，以有限定义无限
  - 是一种比迭代循环更简单、更好用的结构，但时间复杂性和空间复杂性比循环结构高
    - ✓“递”环节需保护现场，开辟运行资源
    - ✓“归”环节需要回收资源
    - ✓占用大量的时间和空间，递归的深度受到限制
  - 必须有明确的递归结束条件
- 适用于当问题需要“后进先出”的操作
  - 树的遍历、图的深度优先搜索、汉诺塔、八皇后…

5!

$$n!=\begin{cases} n!=1 & (n=0,1) \\ n\cdot(n-1)! & (n>1) \end{cases}$$




```
#include<stdio.h>
int  main( )
{ int fac(int n);
  int y, n=5;
  y=fac(n);
  printf("%d!=%d\n", n, y);
  return 0;
}
```

```
int fac(int n)
{
 int f;
 if(n<0)
     printf("n<0, error!");
 else if(n==0||n==1)
              f=1;
       else  f=fac(n-1)*n;
   return(f);
}
```

$$\sum_{n=1}^{100} n \qquad \sum n = \begin{cases} n=1 & (n=1) \\ n+\sum(n-1) & (n>1) \end{cases}$$

```
1+2+…+100=5050
```

```
#include<stdio.h>
int  main( )
{ int fac(int n);
  int y, n=100;
  y=fac(n);
  printf("1+2+…+%d!=%d\n", n, y);
  return 0;
}
```

```
int fac(int n)
{
 int f;
 if(n<=0)
     printf("n<0, error!");
 else if(n==1)
             f=1;
       else  f=fac(n-1)+n;
   return(f);
}
```

枚举技术

Enumerate，也称遍历技术、穷举技术、暴力技术

- 基本思想：选定范围、逐次排查
  - 选定范围
    - ✓明确问题的解的范围，将所有可能解全部列举出来
  - 逐次排查
    - ✓根据约束条件逐个筛选满足条件的解
    - ✓模式：区间枚举、递增枚举

- 适用于循环+判断结构

  例：取一个值各位上的数字、求素数、因式分解…

- 使用条件
  - 能够预先确定解的范围，并能以合适的方法列举
  - 能够对问题的约束条件进行精确描述
- 性能特点
  - 比较直观，易于理解，正确性容易证明
  - 需要一一列举各种可能性，效率较低

例：中国古代数学“鸡兔同笼”问题

“今有鸡兔同笼，上有三十五头，下有九十四足，问鸡兔各几何”（《孙子算经》）

```
#include<stdio.h>
void  main( ) {
  int i, j;
  for(i=0;i<=35; i++)
          for(j=0;j<=35; j++)
                  if(i+j==35 && 4*i+2*j==94)
                          printf("\n Rabbit=%d  chichken=%d\n\n", i, j);
}
```

```
Rabbit=12  chichken=23
```

思考：百钱买百鸡“公鸡每只5元，母鸡每只3元，三只小鸡1元，用100元买100只鸡，问公鸡、母鸡、小鸡各多少？）

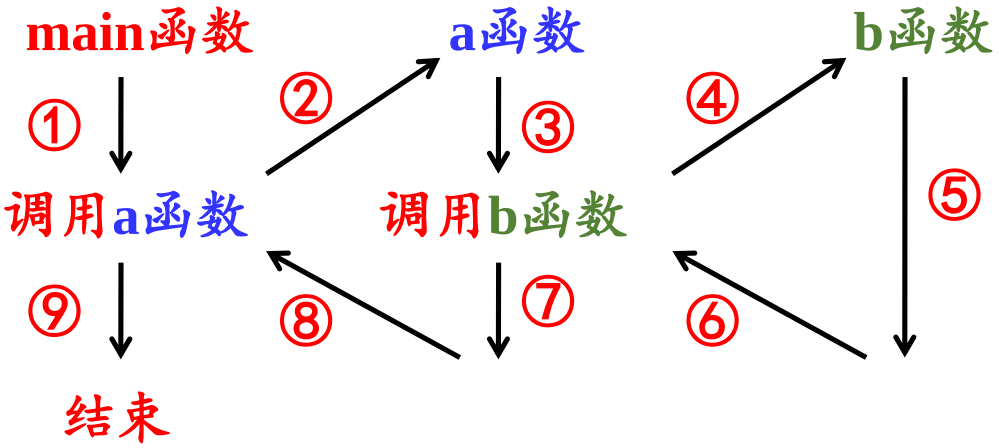
嵌套技术

- 基本思想：内外有别、逐层推进
  - 内外有别
    - ✓外层结构包含了对内层的调用或完整的内层结构，不会出现结构之间相互包含的情形
  - 逐层推进
    - ✓由外向内、先内后外

- 性能特点
  - 自顶向下、逐步细化策略的应用
- 应用于
  - 函数嵌套调用
  - 结构体类型的嵌套定义
  - 三种基本结构之间的相互嵌套
  - …

函数的嵌套调用

- C语言的函数定义是互相平行、独立的，即即函数不能嵌套定义，但可嵌套调用函数，即调用一个函数的过程中，又可调用另一个函数



结构体类型的嵌套定义

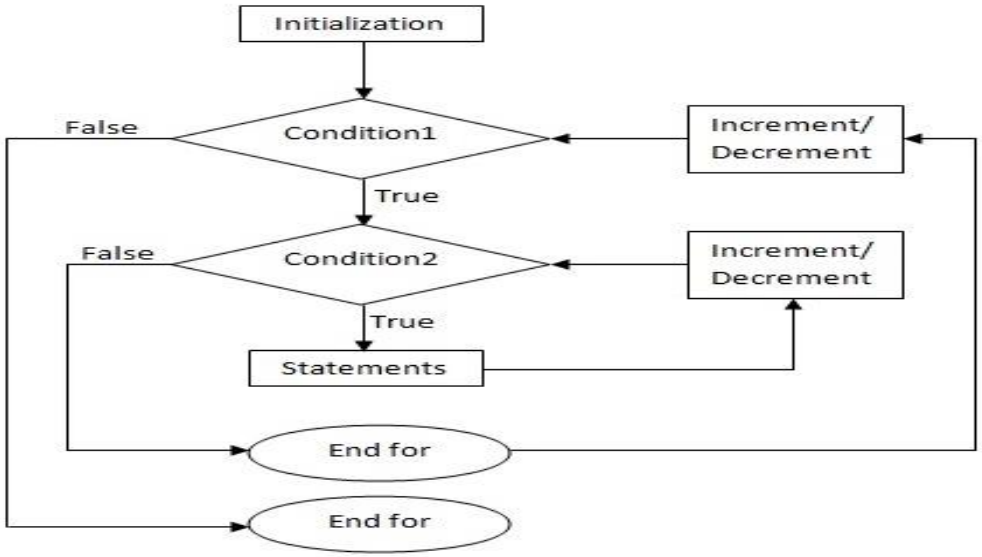
- C语言中当定义结构体类型时，其中的成员又是一个结构体类型，或成员指向该种类型的指针

```
struct Date
{ int month;  int day;  int year; };

struct Stu
{   int num;  char name[20];
    char sex;  int age;
    struct Date birthday;
    char addr[30];
};
```

```
struct  student
{
    int  num;          // 学号
    int  score;        // 成绩
    struct  student  *next;
                // 下一结点的地址
};
```

基本结构之间的相互嵌套

- 是指结构化程序设计的三种基本结构中包含另外一个完整的结构，形成多层的结构



Homework

- 实践

  《学习辅导》：p255　实验2

- 作业

  《教材》：P35　7、8