



程序和流程控制

C语言程序的书写格式有规律吗？

设计C语言程序应从哪些方面思考问题？ C语言程序的基本结构有哪些？

编写一些简单的C语言程序有规律可循吗？

如何培养严谨的思维和良好的习惯？

目录 content

3.1 C语言程序的版式及语句

3.2 结构化程序设计和流程控制

3.3 if语句

3.4 switch多分支选择语句

3.5 循环控制

3.6 辅助控制语句

3.7 典型程序编写方法举例

3.8 本章小结

本章主要从编程的思路和理解程序的结构框架出发，来说明程序的构架和格式(程序的版式)。

例3.1 求二整数之和的程序

```
#include <stdio.h>    //预处理
void main()           //函数定义
{
    int a,b;           //变量说明
    int sum;
    scanf("%d%d",&a,&b); // 数据输入
    sum = a + b;        // 执行部分
    printf("sum=%d",sum); // 信息输出
}
```

这是一个典型的只包含单个函数（即main()）的程序

例3.1由**七块**组成：

- (1) 注释部分
- (2) 预处理块、全局变量说明、函数声明等
- (3) 函数定义部分
- (4) 变量说明部分
- (5) 数据输入部分
- (6) 执行部分
- (7) 信息输出部分

例3.2

求二整数之和的程序

如例3.1程序的执行部分采用函数调用则例3.1程序可修改为:

```
#include <stdio.h>           //预处理
int add(int x,int y);         //函数声明

void main( )                  //函数定义
{
    int a,b;                  //变量说明
    int sum;

    scanf("%d%d",&a,&b); //数据输入

    sum = add(a, b);           //执行部分

    printf("sum=%d",sum); //信息输出
}
```

```
/*求和函数，输入参数为二整数，
   返回值为其和*/
int add(int x, int y) //函数定义
{
    int z;              // 变量说明

    z = x + y;          //执行部分

    return z;           //返回结果
}
```

输入：7 8 (CR)
运行结果为：
sum=15

语句是程序的基本元素，程序中的各功能部分都是由一定含义的语句组成的，换句话说，语句是一个完整程序的基本组成部分。

C语句的特点是以**分号为结束符**。

例如: `x= 10` `/*不是语句*/`

`y= 7 ;` `/*分号结束构成语句*/`

根据语句的作用可以把语句分成**说明语句**和**执行语句**两大类。

说明语句： 用来对程序中所使用的各种类型变量及属性进行说明，按其所以起作用有时也称为定义语句。

说明语句的格式为：

<存储类型> 数据类型 变量名列表；

说明语句也可以初始化，如：

```
char ch = 'H' ;
```

```
unsigned long y = 356847412 L;
```

执行语句一般包含四大类，分别是：

表达式语句(包括空语句)

复合语句

流程控制语句

辅助控制语句

表达式语句：任何一个表达式加上一个分号就是一条语句，一般的格式为：

表达式; //表达式语句

只有分号而没有表达式的语句就叫**空语句**；空语句格式为：

; //空语句

复合语句：将若干语句用括号{ }括起来就构成了复合语句，
复合语句在语法上相当于一个语句。

复合语句的一般格式为：

```
{  
    语句1;  
    语句2;  
    .....  
}
```

本节包括两部分的内容：结构化程序设计和 C 语言的流程控制语句和辅助控制语句。

任何结构化程序都可以用三种基本结构表示：

- 顺序结构
- 选择结构
- 循环结构

- 按语句顺序依次执行

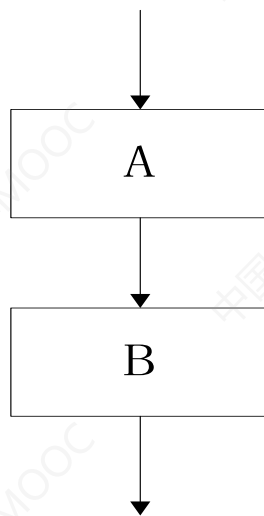
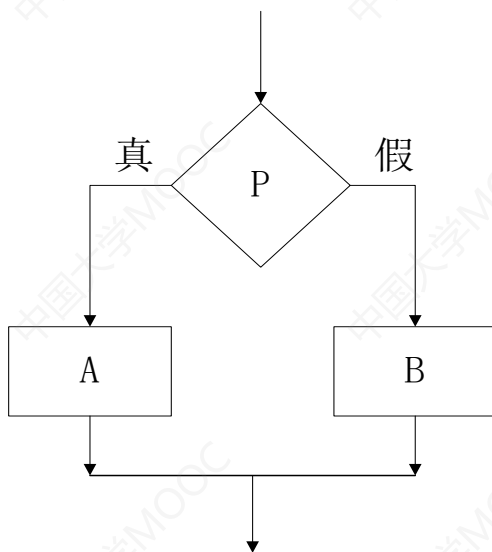


图3.1 顺序结构示意图

- 根据给定的条件进行判断，由判断结果决定执行两支或多支程序段中的一支：



由两分支选择结构可以派生出另一种基本结构，多分支选择结构。

- 在给定条件成立的情况下，反复执行某个程序段。有**当型循环结构**和**直到型循环结构**。

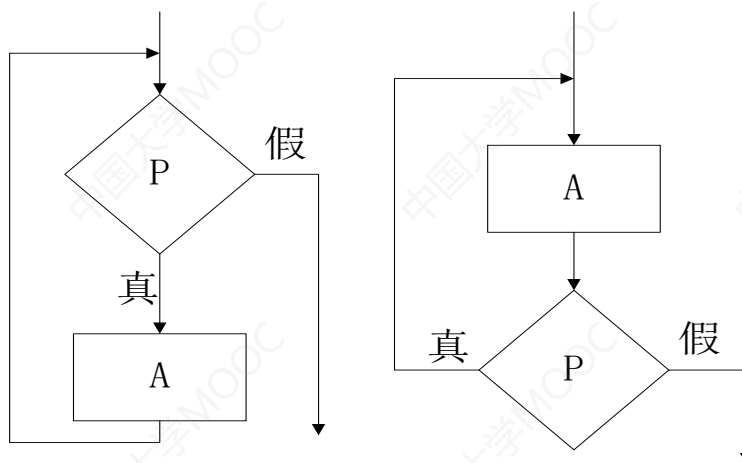


图3.3 循环结构示意图

根据结构化程序结构的基本思想，程序的执行过程可以用上述的流程结构和结构的嵌套表示出来，而C语言又为这种流程结构提供了对应的流程控制语句，这样就能非常方便地将程序的执行过程用C语言描述出来，程序执行过程的C语言描述就是我们所编写的程序的主体。

流程控制语句

选择

if else
switch case

循环

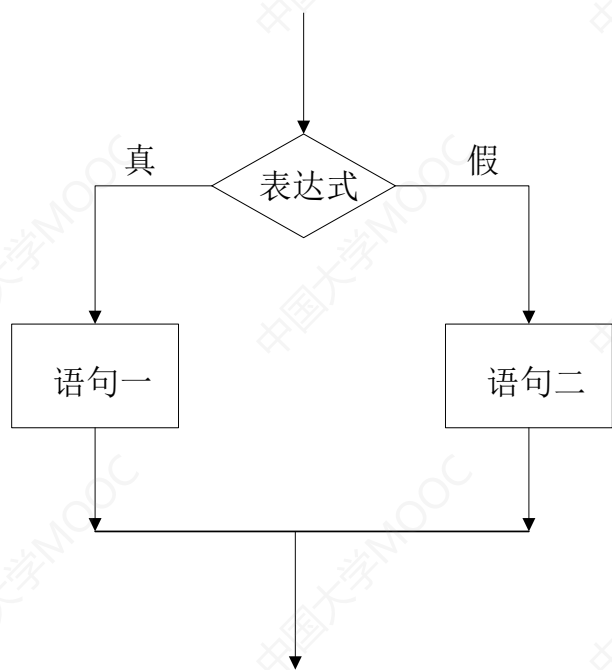
for
while
do while

辅助控制语句

控制语句

break
continue
goto
return

- **if-else条件分支语句**的标准使用形式，它的流程和语句形式如下：



```
if (表达式)
{
    语句一;
}
else
{
    语句二;
}
```

示例：

```
if(a>=0)
{
    printf("come on ,baby!");
}
else
{
    printf("go away!");
}
```

图3.5 标准if-else的流程和语句

建议结构化语句的执行体部分都采用复合语句。

- if后面圆括号中的表达式一般是关系表达式或逻辑表达式，它表示分支的条件。
- 在C语言程序中，下面两种方法经常使用：
if(x) 等价于 if(x!=0)
if(!x) 等价于 if(x==0)
- 如变量x为float等实型变量，则与零值比较的标准if语句如下：
if (fabs(x)<= 1e-6)

3.3.1

if 语句程序注意事项2

- 程序中有时会遇到if/else/return的组合，应该将如下不良风格的程序：

```
if (condition)
    return x;
    return y;
```

改写为

```
if (condition)
{
    return x;
}
else
{
    return y;
}
```

或者改写为

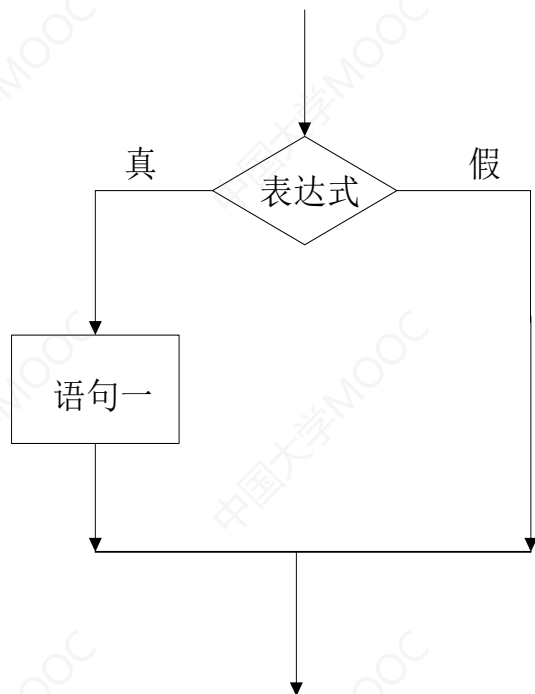
```
return (condition ? x : y);
```

最简

3.3.1

if 语句程序注意事项3

- if分支是if-else分支的缺省情况，即缺省else时的条件分支。



```
if (表达式)
{
    语句一;
}
```

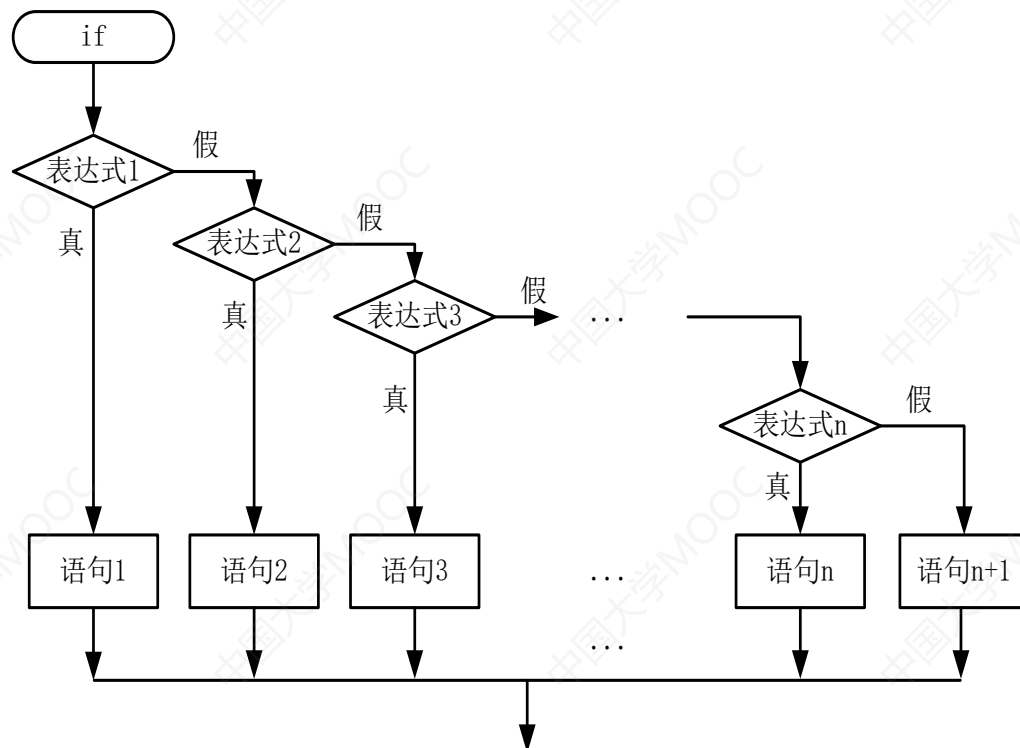
图3.6 缺省else时的条件分支的流程和语句

例如:

```
if ( i<100)
{
    i++;
}
```

多条件分支下的流程控制表示的流程结构如下:

```
if(表达式1)
{
    语句1;
}
else if(表达式2)
{
    语句2;
}
else if(...)
...
else
{
    语句n;
}
```



例3.3

if--else if 示例

为了给某班学生的一次考试成绩分等级，其中*i*表示学生成绩，*grade*表示等级。90分以上的为A，70分到90分之间的为B，60分到70分之间的为C，60分以下的得D。

```
.....  
if (i>=90)  
{  
    grade= 'A' ;  
}  
else if (i>=70)  
{  
    grade='B' ;  
}  
else if (i>=60)  
{  
    grade='C' ;  
}  
else  
{  
    grade='D' ;  
}
```

3.4

switch——多分支选择语句

switch也是分支选择语句，它可以是**多分支选择**，而if语句只有**两个分支**可供选择。

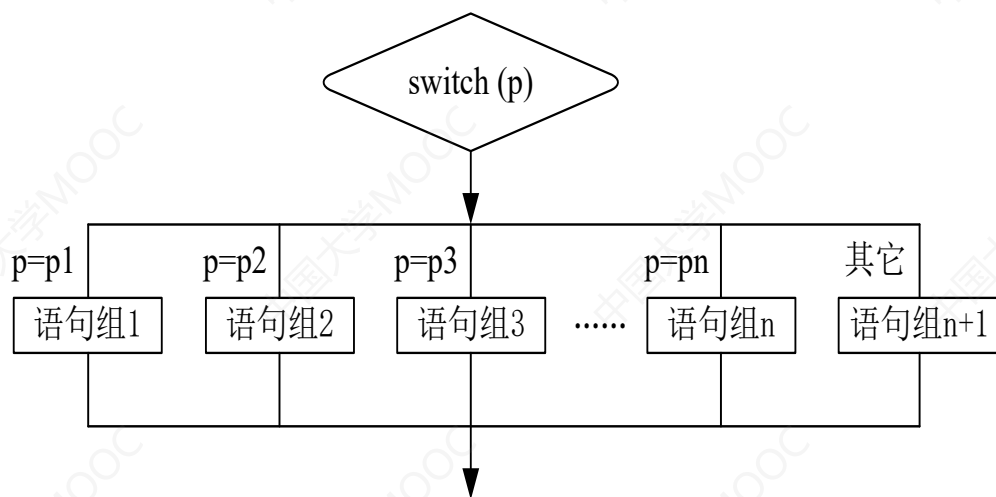


图3.8 switch语句的流程控制

```
switch(表达式)
{
    case 判断值1:
        语句组1;
        break;
    case 判断值2:
        语句组2;
        break;
    .....
    case 判断值 n:
        语句组n;
        break;
    default:
        语句组n+1;
        break;
}
```

switch语句示意程序

例3.5 编一示意性的菜单处理程序，按下一功能键，执行响应的功能处理。

```
#define ESC 0x011b
#define F1 0x3b00
//F1键的键值为0x3b00
#define F2 0x3c00
#define F3 0x3d00
#define F4 0x3e00
#define F5 0x3f00
#define F6 ....
#include <stdio.h>
#include <bios.h>

void main( )
{
    unsigned int key_value;


    key_value = bioskey(0);
```

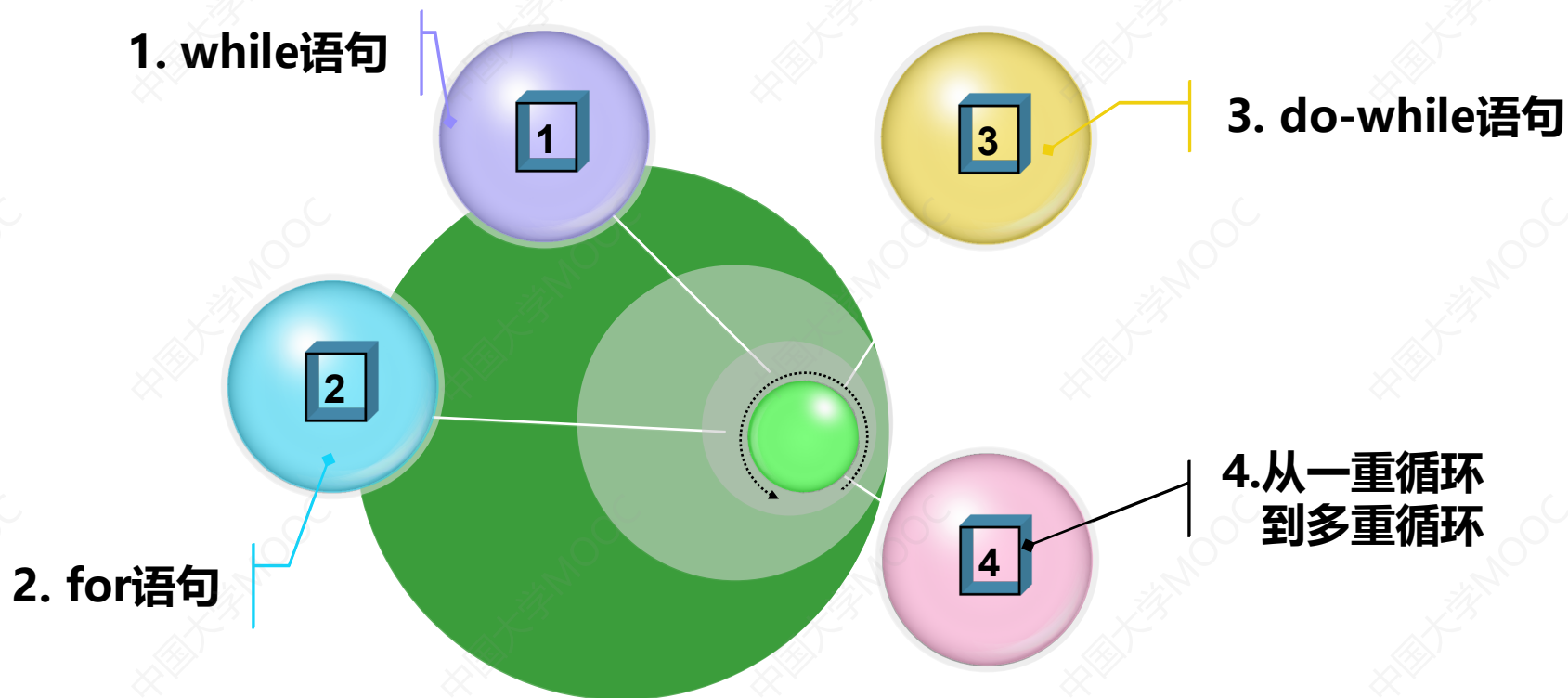
```
switch (key_value)
{
    case F1: F1功能处理程序;
            break;
    case F2: F2功能处理程序;
            break;
    case F3: F3功能处理程序;
            break;
    case F4: F4功能处理程序;
            break;
    case F5: F5功能处理程序;
            break;
    case F6: F6功能处理程序;
            break;
    .....
    default: 相应处理程序;
            break;
}
```

选择与key_value
值相对应的处理
程序

未找到对应值时，
进入该处处理程序

关于switch语句，**注意**以下几点：

- switch ()后面圆括号中的表达式要求结果是整数(整形变量)，各个case的判断值要求是整型常量。
- 各个case和default及其下面的语句组的顺序是任意的，但各个case后面的判断值必须是不同的值。
-  多个分支语句组的break语句起着退出switch-case结构的作用，若无此语句，程序将顺序执行下一个case语句组。
- 当表达式的结果值与所有的case的判断值都不一致时，如果程序中存在default语句，则程序执行default部分的语句组；如果程序中没有default语句，程序则退出switch case结构。



- **循环结构**是在给定条件时，反复执行某个程序段，反复执行的程序叫循环体。
- C语言有**三种循环流程控制**。
while循环，for循环，do-while循环。
- while 循环的程序流程和程序形式：

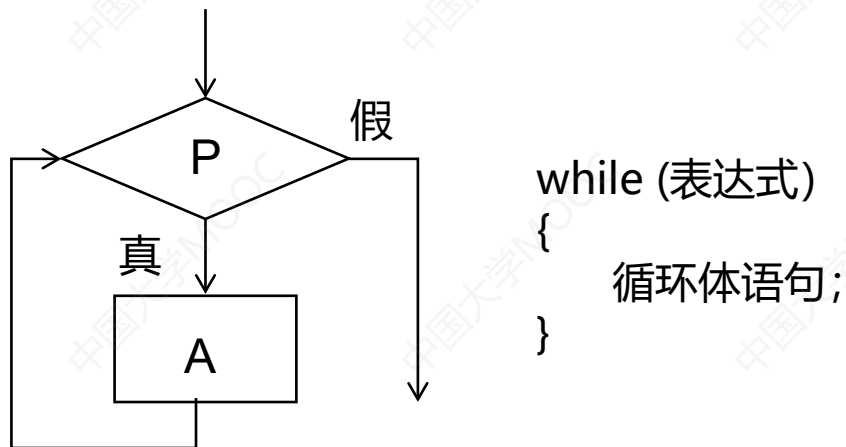


图3.9 while语句的流程及语句形式

例3.7

用 while 循环语句编写一程序求1到100之间所有整数之和

```
#include <stdio.h>
void main( )
{
    int i;
    int sum;
    ● sum = 0;
    ● i = 1;
    ● while ( i <= 100)
    {
        sum = sum + i;
        i ++;
    }
    printf("sum = %d\n", sum);
}
```

赋初值

当 i > 100 时跳出
循环

运行
结果

sum = 5050

1. while循环的表达式是循环进行的条件。用作循环条件的表达式中一般至少包括一个能够改变表达式的变量，这个变量称为循环变量。
2. 当条件表达式的值为真(非零)时，执行循环体，为假(等于0)则循环结束。
3. 当循环体不需要实现任何功能时，可用空语句作为循环体。
如: `while((ch=getchar()) != ' A') ;`
4. 循环语句应有出口。
5. 对于循环变量的初始化应在while()语句之前进行。
6. while语句中条件表达式的写法与前面if语句中条件表达式的写法基本相似。

for 循环是功能上比while循环更强的一种循环结构

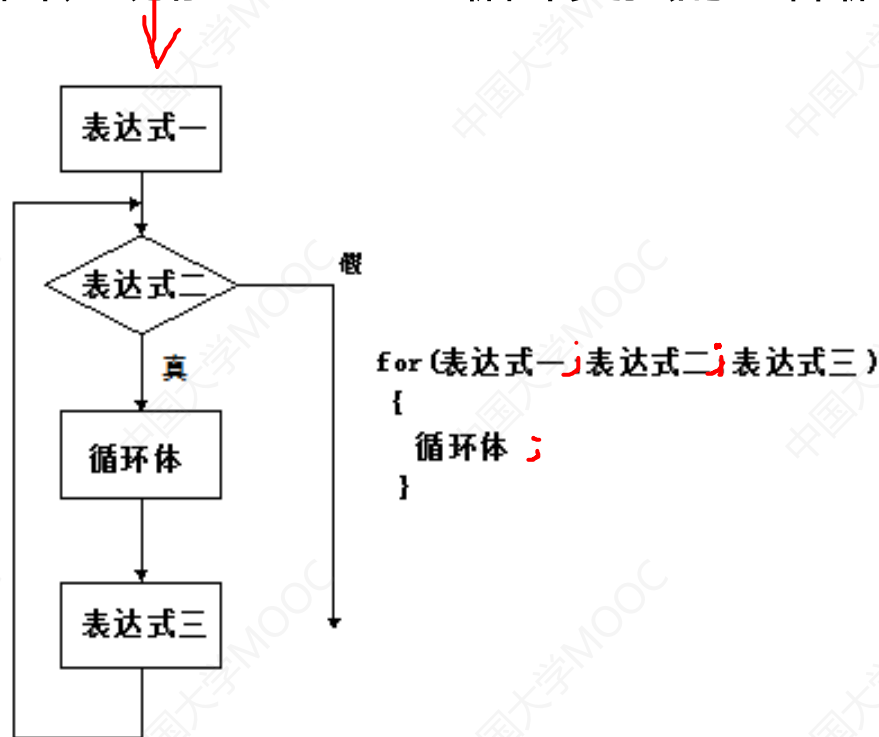


图3.10 for语句的流程和语句形式

- **for循环**通常用于构造"初值、终值、步长"型循环，如：

```
for ( i=0 ; i<100; i+=5)
{
    printf("%d\n",i);
}
```

- for循环的三个表达式起着不同的作用：
 - 表达式1用于进入循环前给某些变量赋初值;
 - 表达式2表明循环的条件;
 - 表达式3用于循环，依次对某些变量的值进行修改。
- 在for循环中，表达式1和表达式3经常是逗号运算表达式

例3.8

用 for 循环语句编写一程序求1到100之间所有整数之和

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

```
void main( )
```

```
{
```

```
    int i;
```

```
    int sum;
```

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

```
    {
```

```
        sum += i;
```

```
    }
```

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

```
}
```

表达式一为逗号表达式，赋初值

表达式二：循环条件

表达式三：循环步长

运行结果

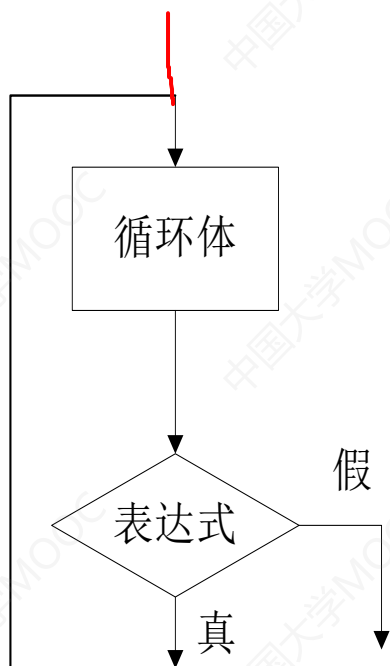
sum =5050

- 表达式1, 表达式2和表达式3可以全部或部分省掉, 但是分号不能省, 相当于永真条件 (条件永远成立), 即for(; ;)等同于 for(; 1 ;), 此种情况下, 必须在循环体中使用break来控制循环的结束。
- 循环体也可以为空语句, 如:

```
for( i=0 ; i<10000 ; i++ );
```

```
或for( i=0 ; i<10000 ; i++ ) { }
```
- 建议for语句的循环控制变量的取值采用"半开半闭区间"写法。

- do-while循环程序流程和程序形式为:



```
do  
{  
    循环体;  
}  
while(表达式);
```

例如：下面的两个语句显示: 0 1 2 3 4

```
int i=0;  
do  
{  
    printf("%3d", i++);  
}while(i<5);
```



关于do-while语句，**注意以下几点：**

- do-while循环体类似于while循环，不同之处是，它们执行循环体与计算表达式的先后顺序不同。
- 能用while循环和for循环描述的程序大多数情况下都能用do-while循环描述。
- 能用do-while循环描述的程序一定能用while循环和for循环描述。

例3.9

利用do - while循环编写程序
求1到100之间所有整数之和(类似例3.7和例3.8)

```
#include <stdio.h>
void main( )
{
    int i;
    int sum;

    ● sum = 0;
      i = 1;
    do
    {
        sum = sum + i;
        i++;
    ● } while ( i<=100);
      printf("sum = %d \n", sum);
    }
```

赋初值

当 i>100时跳出
循环

运行
结果

sum = 5050

- 先考虑while循环，在一重循环中， while循环语句的格式如下：

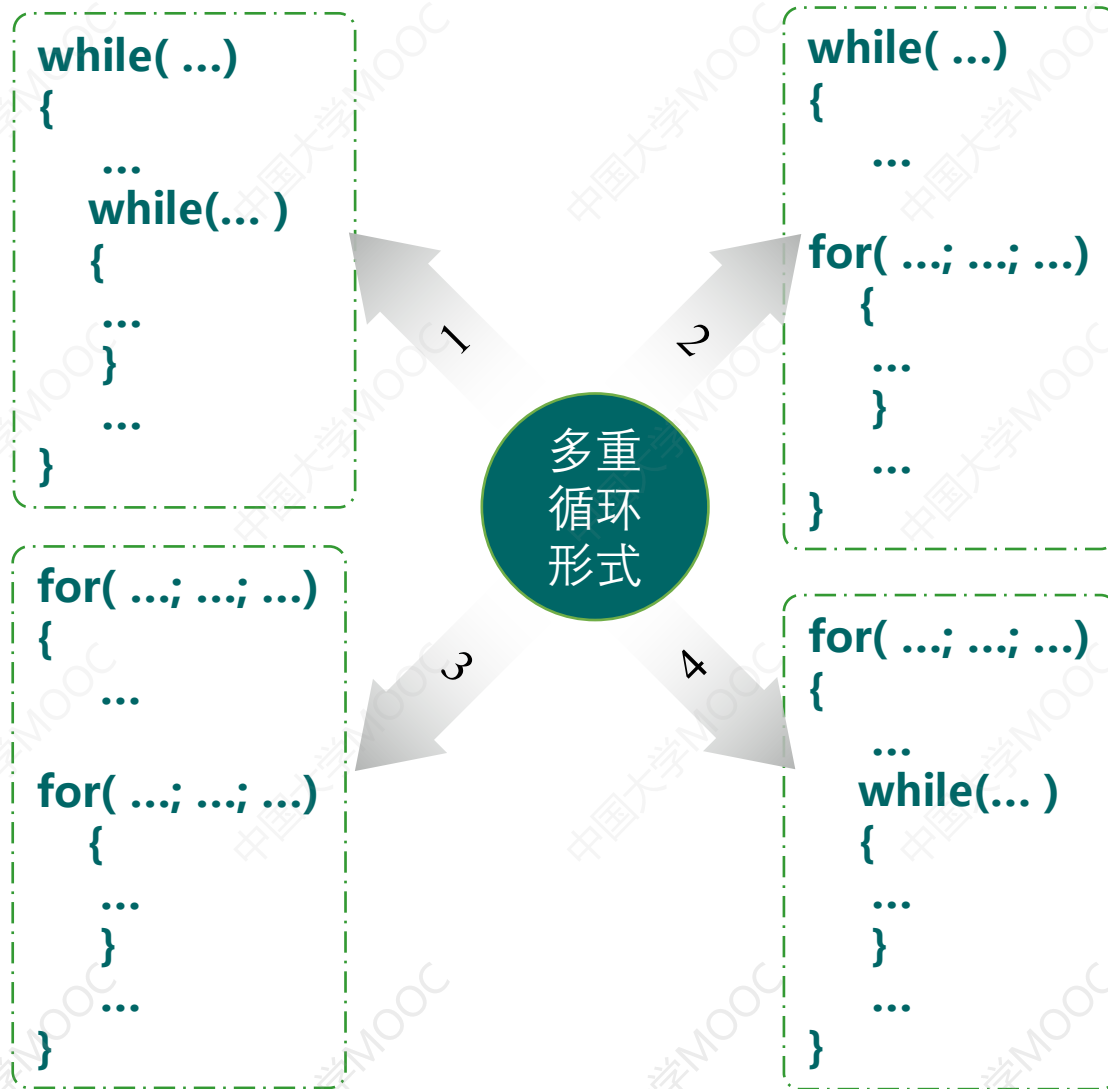
```
while(条件表达式)
{
    循环体部分;
}
```

- 当循环体部分的功能无法用顺序和选择结构而必须用循环结构来实现时，上述的结构就变成如下的形式：

```
while(条件表达式1)
{
    while(条件表达式2)
    {
        循环体部分2;
    }
}
```

这种结构的形式就是二重循环，可用来解决比一重循环更复杂的问题

在多重循环中，内层循环可以认为是外层循环的循环体，依此类推。





- 注意循环控制变量赋初值的位置。
- **内外循环变量不应该同名。**
- 应正确地书写内外循环体，需要在内循环执行的所有语句必须用{ }括起来组成复合语句作为内层循环体;属于外循环的语句应放在内层循环体之外，外循环之中。
- 不应该在循环中执行的操作应放在最外层循环进入之前或最外层循环结束后。

例3.10

编程显示输出如下所示的三角形的程序

```
*  
***  
*****  
*****  
*****  
*****
```

```
#include <stdio.h>  
void main( )  
{  
    int i,j;  
    for ( i = 0 ; i < 6 ; i ++)  
    {  
        printf("\n");  
        for(j = 0 ; j < 5-i ; j ++)  
        {  
            printf(" ");  
        }  
        for(j=0 ; j<2*i+1 ; j++)  
        {  
            printf("*");  
        }  
    }  
}
```

3.6

辅助控制语句

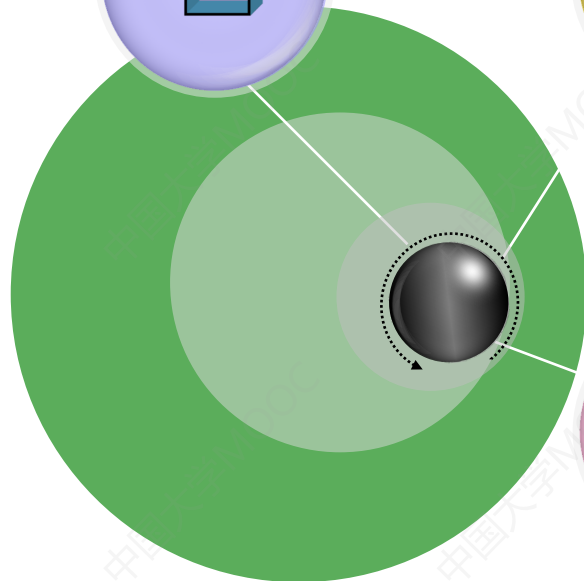
1. break语句



2. continue语句



3. goto 语句
和标号



不能用于循环体语句和switch语句之外的任何其它语句。

- 可以使流程跳出switch结构，继续执行switch下面的语句。
- 可以用来从循环体内跳出循环体，既结束当前循环，执行循环下面的语句。



注意：break语句只能跳出一层循环。

例3.12

编一示意性的菜单处理程序，要求按下一功能键，执行响应的功能处理，重复执行直到按ESC键退出。

将例3.5进行简单修改得到

```
#define ESC 0x11b;
#define F1 0x3b00
//F1键的键值为0x3b00
#define F2 0x3c00
#define F3 0x3d00
#define F4 0x3e00
#define F5 0x3f00
#define F6 0x4000
.....
#include <stdio.h>
#include <bios.h>
void main( )
{
    unsigned int key_value;
    while(1)
    {
        key_value = bioskey(0);
        if(key_value == ESC)
            break;
```

跳出while循环

```
switch (key_value)
{ case F1: F1功能处理程序;
      break;
  case F2: F2功能处理程序;
      break;
  case F3: F3功能处理程序;
      break;
  case F4: F4功能处理程序;
      break;
  case F5: F5功能处理程序;
      break;
  case F6: F6功能处理程序;
      break;
  .....
  default: 相应处理程序;
      break;
}
```

跳出switch结构

跳出switch结构

跳出switch结构

跳出switch结构

跳出switch结构

跳出switch结构

跳出switch结构

- **功能：**结束本次循环，即跳过循环体尚未执行的语句，接着进行下一次是否执行循环的判定。

注意：continue语句结束本次循环。

```
void main( )
{
    int n;
    for(n=0;n<=100;n++)
    {
        if(n%5!=0)
            continue;
        printf("%d\t",n);
    }
}
```

在某些场合使用
continue语句可以提高
整个程序的效率

```
void main( )
{
    int n;
    for(n=0;n<=100;n++)
    {
        if(n%5==0)
            printf("%d\t",n);
    }
}
```

程序功能：把0 - 100之间
能被5整除的数输出

```
#include<stdio.h>
#include <math.h>
#define PI 3.1415926
void main( )
{
    double r, area;
    while (1)
    {
        printf("input the radius:");
        scanf("%lf", &r);
        ● if (fabs(r) < 1e-5)
            break;
        else if (r < 0.0)
        {
            printf("the input is error\n");
            continue;
        }
        area = PI * r * r;
        printf("the area is:%lf\n", area);
    }
}
```

确定输入圆半径值有意义



输入一个圆的半径，输出圆的面积。
现在我们对这个程序进行改进，要求：
(1).允许反复的输入半径，计算并显示圆的面积，直到输入的半径是0时为止(输入0是终止程序运行的信号);
(2).对输入的半径进行检查，若发现是负数将提示操作者重新输入。

continue语句只是结束本次循环，而不是中止整个循环，而**break语句**则是结束整个循环过程，不再判断执行循环的条件是否成立。

- 程序中使用goto语句时要求和标号配合，一般形式：

goto 标号;

.....

标号: 语句;

- **功能：**把程序控制转移到标号指定的语句处。既执行goto语句之后，程序从指定标号处的语句继续执行。
- 注意: goto语句常用的用法是用它退出多重循环。
- **！ goto语句为非结构化语句，我们不提倡使用！**

本章我们强调了程序的风格和结构的规范化，但是当我们面对一个较为复杂的编程问题时，是不可能立即编写出风格和结构最佳的程序的，一般的方法是采用自顶向下逐步求精的模块化，结构化的方法进行分析和设计，把一个复杂问题变成若干便于实现的小问题。

本节将介绍三种处理简单问题的程序设计典型模板。

- 多项式累计法
- 显式条件试数法
- 隐式条件处理法

典型问题1

例3.15 求序列：1, 3, 5, 7, 9.....的前二十项之和。

```
#include<stdio.h>
void main( )
{
    int i;
    int sum, t; //sum代表和, t代表某项

    sum = 0;
    t = 1;

    for (i=0 ; i<20 ; i++)
    {
        sum += t;
        t += 2;
    }

    printf(" sum = %d",sum);
}
```

将前二十项的值
进行累加

运行结果

sum =400

例3.15

分析1

- 分析上面的程序我们不难看出该程序的结构大致如下:

头文件部分

```
void main( )
```

```
{
```

```
    变量说明部分;
```

```
    初始化 ( 和清零, 项变量初始化第一项)
```

```
    循环( 根据条件决定)
```

```
{
```

```
    累加一项;
```

```
    根据本项计算下一项;
```

```
}
```

```
    输出结果;
```

```
}
```

请回忆“七块”理解法,
找出对应此结构的位置

例3.16

求序列：1!, 2!, 3!, 4!.....的前八项之和.

```
#include<stdio.h>
void main( )
{
    int i;
    long sum, t;
    //sum代表和, t代表某项
    sum = 0;
    t = 1;

    for (i=1 ; i<=8 ; i++)
    {
        sum += t;
        t *= (i+1);

        printf(" sum = %ld",sum);
    }
```

运行结果

sum =46233

计算下一项阶乘的值

比较[例3.15]和[例3.16]，区别仅仅在于以下几点：

(1).变量说明不一样

(2).循环的条件不一样

例3.17

求序列： $\frac{1}{2}, \frac{3}{4}, \frac{5}{8}, \frac{7}{16}, \frac{9}{32}, \dots$ ，所有大于等于0.000001的数据项之和，显示输出计算的结果。

```
#include <stdio.h>
#include <math.h>
void main( )
{
    float sum, a, b;
    //sum代表和, a为分子, b为分母
    sum = 0;

    a = 1;    // 分子赋初值
    b = 2;    // 分母赋初值

    while (a/ b >= 1e-6)
    {
        sum = sum + a/ b; //累加一项
        a = a+2;    //求下一项的分子
        b = b*2;    //求下一项的分母
    }

    printf(" sum = %f",sum);
}
```

运行结果

sum =2.999999

浮点数的舍入误差造成的现象

例3.18

```
#include <stdio.h>
#include <math.h>
void main( )
{
    int i;
    float x, sum, a, b; //sum代表和, a为分子, b为分母
    char s;
    printf("please input x:");
    scanf("%f", &x);
    s=1;
    sum = 0;
    a = x;    // 分子赋初值
    b = 1;    // 分母赋初值
    for (i=1; a/ b>=1e-6; i++)
    {
        sum = sum + s* a/ b; //累加一项
        a = a*x*x;    //求下一项的分子
        b = b*2*i*(2*i+1); //求下一项的分母
        s*=-1;
    }
    printf("sum = %f\n",sum);
}
```

计算 $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \dots\dots$,
并使最后一项的绝对值小于为止。

运行结果

please input x:2
sum = 0.9092974

例3.19

求100 - 999之间所有的水仙花数，所谓水仙花数就是说数的百位、十位和个位数的立方和恰好等于它本身。

```
#include<stdio.h>
void main( )
{
    int i,a,b,c;
    for (i=100;i<=999; i++)
    {
        a = i/100 ;      //求百位数
        b =( i- a*100)/10; //求十位数
        c = i%10;        //求个位数

        if( a*a*a + b*b*b + c*c*c == i)
        {
            printf(" %6d", i );
        }
    }
}
```

判断是否为水仙花数

运行结果

153 370 371 407

程序的结构大致如下:

头文件部分

```
void main( )
```

```
{
```

变量说明部分;

初始化 (可缺省);

循环(根据条件决定)

```
{
```

先期处理(可缺省);

根据条件判断输出所得结果(也可能是包含循环的程序结构);

```
}
```

```
}
```

- 这种循环条件加判断的编程方法我们这里把它叫做**试数法**。

例3.20

题目： 小学生的加法算式

$$\begin{array}{r} a \ b \ c \\ + \ c \ b \ a \\ \hline 1 \ 3 \ 3 \ 3 \end{array}$$

a,b,c为一位数,试编程求所有可能a,b,c的值

a=4,b=1,c=9
a=5,b=1,c=8
a=6,b=1,c=7
a=7,b=1,c=6
a=8,b=1,c=5
a=9,b=1,c=4

运行结果

```
#include<stdio.h>
void main( )
{
    int i;
    int a, b ,c;

    for ( i=100 ; i<=999; i++)
    {
        a = i/100 ;

        b =( i- a*100)/10;

        c = i%10;

        if( 1333 == a*100+b*10 + c +
            c*100 + b*10 + a)
        {
            printf("a= %d b=%d  c=%d\n", a,b,c );
        }
    }
}
```

例3.21

求已知两个正整数的最大公约数。

将例3.21与例3.20进行比较

please input a ,b:6 4
the max =2

运行结果

```
#include<stdio.h>
void main( )
{
    int i;
    int a, b ;
    printf("please input a ,b:");
    scanf("%d%d", &a,&b);
    for ( i= a<b ? a:b ; i>0 ; i--)
        // i初值为a,b中的较小值
    {
        if( a%i ==0 && b%i==0)
        {
            printf("the max = %d ", i );
            break;
        }
    }
}
```

例3.22

题目：百钱百鸡问题，用100元钱买100只鸡，其中母鸡每只3元，公鸡每只2元，小鸡1元3只，且每种鸡至少买一只，试编一程序列出所有可能的购买方案。

运行结果

母鸡数:5 公鸡数:32 小鸡数:63
母鸡数:10 公鸡数:24 小鸡数:66
母鸡数:15 公鸡数:16 小鸡数:69
母鸡数:20 公鸡数:8 小鸡数:72

```
#include <stdio.h>
void main()
{
    int a, b, c;
    //a,b,c分别代表母鸡,公鸡和小鸡数
    for ( a=1; a<=98;a++)
    {
        for (b=1;b<=98;b++)
        {
            for(c=1;c<=98;c++)
            {
                if ((a+b+c==100) &&(a*3+b*2
                    +c/3==100)&&( c%3==0))
                    printf("母鸡数:%d 公鸡数:%d 小鸡数:%d\n",a,b,c);
            }
        }
    }
}
```

3.7

典型问题3

•**例3.25:** 编一程序实现一个最简单的计算器的功能，如输入3+5回车显示3+5=8;输错就退出(输入的不是加减乘除的运算就算错)

```
#include <stdio.h>
#include <math.h>
void main()
{
    float a, b, s;
    char op;
    while (1)
    {
        scanf("%f%c%f", &a, &op, &b);
        if ((op != '+') && (op != '-') && (op != '*') \
            && (op != '/'))
            break;
        switch (op)
        {
            case '+':
```

根据输入的运算符进行相应的运算

```
                printf ("%f+%f=%f", a, b, a + b);
                break;
```

```
            case '-':
                printf("%f-%f=%f", a, b, a - b);
                break;

            case '*':
                printf("%f * %f=%f", a, b, a * b);
                break;

            case '/':
                if (fabs(b) < 1e-6)
                    printf("除法错误 确保除数不为零");
                printf("%f / %f = %f", a, b, a / b);
                break;
        }
    }
}
```

运行结果

12.3+45.6 //输入:
12.30000+45.600000=57.900000

例3.26

•**题目：** 编程序为小学生出一套最简单的整数(最大不超过100)加减乘运算的试题，试题共十道题，每道题随机产生，产生后学生立即给答案，计算机立即判断正确和错误，十道题做完给出成绩。

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
void main( )
{
    int a , b, s;
    char op ;
    int score=0;
    randomize(); // 随机数发生器初始化
    for (i=0; i<10; i++)
    {
        a = random(101);
        //随机产生一个0—100的整数
        b = random(101);
        //随机产生一个0—100的整数
        op = random(3);
        //随机产生一个0—2的整数
        if(op == 0) //0代表加
        {
            printf("%d + %d = ? ",a ,b);
            scanf("%d", &s);
```

```
        if (s==a+b)
        {
            printf("true");
            score =score+10;
        }
        else printf("false");
    }
    else if(op == 1) // 1代表减
    {
        printf("%d - %d = ? ",a ,b);
        scanf("%d", &s);
        if (s==a-b)
        {
            printf("true");
            score =score+10;
        }
        else printf("false");
    }
    else if(op == 2)
        //2代表乘
```

```
    {
        printf("%d * %d = ? ",a ,b);
        scanf("%d", &s);
        printf ("score =%d ", score);
    }
    if (s==a*b)
    {
        printf("true");
        score =score+10;
    }
    else printf("false");
}
}
```

- “七块” 理解法（哪七块？）
- 注意培养结构化的思维模式，这对同学们今后的编程都大有好处。
- 学会归类一些典型问题，掌握好典型问题的分析处理方法，套用现有的算法结构可以为我们分析新的问题减少许多工作量。
- 养成良好的编程习惯，主要有如下两点：
 - 1)关于程序风格--培养良好的编程风格是学习程序设计的重要的一环。学会后要一如既往的坚持并保持风格的一致，程序的风格虽然都是些细小的事情，但对程序的质量有着至关重要的关系。
 - 2)关于结构化编程--结构化编程是程序设计的基础，初学者要在掌握基本语法要点的基础上，通过大量的编程实践来熟练掌握条件语句和循环语句的用法。