循环控制结构

在程序设计中需要对重复执行的操作采用循环进行处理。当条件满足时执行循环操作，知道条件不满足时跳出循环。

2.1常见的循环结构有while,do…..while以及for循环。

（1）while语句的一般形式：

while(表达式)

{

语句1;

语句2;

…..语句n;

}

当表达式的值为真或者非零时，执行循环语句。

（2）do….while语句的一般形式：

do

{

语句

}

while(表达式)

do…while语句构成的循环，即先执行循环语句，知道条件不满足时跳出。do…while和while的区别在于：do…while是先执行语句，再判断条件，不管条件是否成立，它至少执行一次循环；而while是先判断条件，再执行语句。

（3）for语句的一般形式：

for(表达式1；表达式2；表达式3)　语句

它的执行过程如下：

1. 计算表达式1。（初始化语句）

2 计算表达式2（循环控制表达式），判断是否为真，假如表达式2为真，执行语句，否则结束循环。

3执行语句。

4计算表达式3（增量表达式）的值，转向步骤2。

先判断，再执行，最后增加数据

5结束循环。

For循环还有许多使用技巧

1. 如果表达式2可以省略，即不判断循环条件，循环无终止地进行下去。

```
for(i=1; ;i++)
sum=sum+i;
```

2. 表达式3也可以省略，但程序设计者应另外设法保证循环能正常结束。

```
for(i=1;i<=100;)
{
        sum=sum+i;
 i++;
}
```

3. 可以省略表达式1和表达式3，只有表达式2。

```
for(;i<=100;)
{
        sum=sum+I;
 i++;
}
```

4. 3个表达式都可省略，即无终止的执行循环。

```
for( ;  ; )
```

5. 表达式1可以是设置循环变量初值的赋值表达式，也可以是与循环变量无关的其他表达式。

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

6. 表达式为一般的关系表达式或逻辑表达式，但也可以是字符表达式，只要其非零，就执行循环体。

```
for(i=0;(c=getchar())!=’\n’;i+=c)
printf(“%d”,c);
```

从上面可以看出，for语句比其它循环语句功能强得多，可以把循环体和一些与循环控制无关的操作当做表达式，程序虽然简洁，但是程序的可读性低。所以我们在使用for语句时，尽量在表达式中写与循环控制有关的表达式。

逗号表达式，从右到左的结合性质，多个表达连在一起构成逗号表达式，也称为顺序求值运算符。

while后面加分号可能会导致死循环的产生。

do while 和while不相同，dowhile最少执行一次，while执行可以是0次；

记录数据的循环方式，

使用for循环，使用while或者do while内部建立一个标识符进行加减处理，

死循环是无法停下的循环结构

循环的嵌套处理

循环嵌套是指在一个循环结构的循环体内包含有一个或多个循环结构。

例6.4如果一个三位整数等于其各位数的立方和，则称这个数为水仙花数。编程找出所有的水仙花数。（法一：扫描各位；法二：扫描100-999，然后提取各位。）

例6.5打印九九乘法表。

循环的嵌套处理

计数控制循环结构：重复处理的此时是已知的。

条件控制循环结构：重复处理的此时是未知的，但是结束的条件是已知的。

循环跳出处理

1. break

break语句的使用场合主要是switch语句和循环结构。

在循环结构中使用break语句，如果执行了break语句，那么就退出循环，接着执行循环结构下面的第一条语句。如果在多重嵌套循环中使用break语句，当执行break语句的时候，退出的是它所在的循环结构，跳出一层循环，对外层循环没有任何影响。

如果循环结构里有switch语句，并且在switch语句中使用了break语句，当执行switch语句中的break语句时，仅退出switch语句，不会退出外面的循环结构。

2. continue

continue语句是这5种结束循环的方式中最特殊的，因为它并没有真的退出循环，而是只结束本次循环体的执行，所以在使用continue的时候要注意这一点。

在for循环中，首先执行表达式1（注意表达式1在整个循环中仅执行一次），接着执行表达式2，如果满足条件，那么执行循环体，如果在循环体中执行了continue语句，那么就跳转到表达式3处执行，接下进行下一次循环，执行表达式2，看是否满足条件；

在while循环中，如果执行了continue语句，那么就直接跳转到表达式处，开始下一次的循环判断；

在do while循环体中如果执行了continue语句，那么就跳转到表达式处进行下一次的循环判断，这一点前面已经验证过了。

3. return语句(可以嵌入在子程序的循环中)

如果在程序中遇到return语句，那么代码就退出该函数的执行，返回到函数的调用处，

如果是main()函数，那么结束整个程序的运行。图3-20为return语句的使用。

如果是在自定义的函数中执行，那么执行return之后就返回到函数的调用处继续往下执行。

一个函数体中可以有多个return语句,但在多个return语句中,被执行的只有一个.当return语句执行时,函数"返回"了.即跳出了该函数体,转而执行别的代码.

每个return语句的return值可以各不相同,当函数返回时,你可以根据return的不同值进行不同的处理.

如:如果返回1表示a>b,返回2表示a<b,返回0表示a=b.

则可定义如下函数:

```
int Comp(int a,int b)
{
if(a>b)return 1;
else if(a<b)return 2;
else
return 0;
```

}

当调用Comp(5,6),由于5<6,所以return 2;执行,此函数返回时,返回值就是2.

4. exit()函数

exit()函数与return语句的最大区别在于，调用exit()函数将会结束当前进程，同时删除子进程所占用的内存空间，把返回信息传给父进程。

注：进程——是一个正在执行的程序；计算机中正在运行的程序实例。

当exit()中的参数为0时，表示正常退出，其他返回值表示非正常退出，执行exit()函数意味着进程结束；而return仅表示调用堆栈的返回，其作用是返回函数值，并且退出当前执行的函数体，返回到函数的调用处。

在main()函数中， return n和exit(n)是等价的。图3-21为exit()函数的使用。

如果执行exit()函数后能够返回到main()函数的调用处，那么可以打印出接下来的信息“调用print()函数之后”，但是运行结果表明在调用exit()函数之后没有任何输出，所以执行exit()函数之后将直接结束整个程序的运行。

6.6 本章扩充内容

（1）结构化程序设计的核心思想

程序应该只有一个入口和一个出口。

结构化程序设计条件：“结构清晰，容易阅读，容易修改，容易验证”作为衡量程序的首要条件。

采用顺序：只有一个出口，只有一个入口，无不可达语句，无死循环

（2）尽量避免使用goto语句,因为它破坏了结构化设计风格,并且容易带来错误的隐患。

..goto语句可以不受限制地转向程序中(同一函数内)的任何地方,使程序流程随意转,向,如果使用不当,不仅有可能造成不可达语句,而且还会造成程序流程的混乱,影响程,序的可读性。当然造成程序流程混乱的根源,其实并非goto语句本身,而在于使用了较多的goto语句标号。例如,在程序中的多个地方,用goto语句转向同一语句标号处,进行相同的错误处理,这种做法不但不会影响程序结构的清晰,相反还会使程序变得更加,简洁。

因此,结构化程序设计规定,尽量不要使用多于一个的goto语句标号,同时只允许在,一个“单入口单出口”的模块内用goto语句向前跳转,不允许回跳。当然,也不能简单地认为避免使用goto语句的程序设计方法就是结构化程序设计方法。结构化程序设计关注的焦点是程摩结构

的好坏,而有无goto语句,并不是程序结构好坏的标志。限制和避免使用goto语句,只是得到结构化程序的一个手段,而不是我们的目的。

(3)采用自顶向下(Top-down)、逐步求精(Stepwise refinement)的模块化程序设,计方法。

（3）常用的程序调试与排错方法,

程序排错的第一步是找出错误的根源,然后才能对症下药。排错是一种技艺,就像解一个谜题或侦破一个杀人谜案,不要忽视程序运行的错误结果,它常常会提供许多发现问题的线索。正确利用逆向思维和推理寻找问题的根源,如果仍然未发现问题的线索,那么可以试一试下面的策略:

(1)采用注释的办法“切掉”一些代码,减少有关的代码区域,调试无误后再“打开”这些注释,即采用分而治之的策略将问题局部化。

(2)采用测试(lereetal Tvsting)方块,增量测试是保持一个可工作的过程，即开发射期先建立一个可运作单元(一已被测试过的可正常工作的代的新代码的触入。由于错误最可能出现在新加入的代码或者与源代码有联系的代码块中。

找到程序的错误后、就考虑如问进行改错,改错时要注意以下3个问题:

(1)不要急于修改缺箱(Bug)、先想考一下、修改相关代码行后会不会引发其他bug

(2)程序中可能潜伏者同一类型的很多bug,这时要柴胜追击,改正所有的,

(3)改错后要立脚进行回归测试(Regresion Test) ,即回过头来,对以前修复过告bug重新进行测试、看该bug是否会重新出现,以避免修复一个bug后又引起其他的bug.

C语言直接提供的类型都有取值范围。当向其赋以超过此范围的数值时,就会产生类型溢出,得到错误的结果,

本章节相关的一些算法

累加、连乘

例6.6求自然对数的底e的近似值。要求计算前20项或使其误差小于10-10。

$$e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots+\frac{1}{n!}$$

判断素数

例6.7

1) 任意输入一个自然数n，判断n是否为素数。
2) 求一段范围内的所有素数。
3) 哥德巴赫猜想（大于等于6的偶数可以写成两个素数的和。）。

穷举法

“穷举法”也称为“枚举法”或“试凑法”，即将可能出现的各种情况一一测试，判断是否满足条件，一般采用循环来实现。

例6.8百元买百鸡问题。假定小鸡每只5角，公鸡每只2元，母鸡每只4元。现在有100元钱要求买100只鸡，编程列出所有可能的购鸡方案。

递推法

“递推法”又称为“迭代法”，其基本思想是把一个复杂的计算过程转化为简单过程的多次重复。每次重复都从旧值的基础上递推出新值，并由新值代替旧值。

例6.9猴子吃桃子。小猴在某天摘桃若干个，当天吃掉一半多一个；第二天吃了剩下的桃子的一半多一个；以后每天都吃尚存桃子的一半多一个，到第7天要吃时只剩下一个，问小猴共摘下了多少个桃子？

求最大值、最小值

在若干个数中求最大值，一般**先假设一个较小的数为最大值的初值**，若无法估计较小的值，则取第一个数为最大值的初值；然后将每一个数与最大值比较，若该数大于最大值，将该数替换为最大值；依次逐一比较。