# 实验2 流程控制实验

## 2.1 实验目的

（1）掌握复合语句、if语句、switch语句的使用，熟练掌握for、while、do-while三种基本的循环控制语句的使用，掌握重复循环技术，了解转移语句与标号语句。

（2）练习循环结构for、while、do-while语句的使用。

（3）练习转移语句和标号语句的使用。

（4）使用Turbo C 2.0集成开发环境中的调试功能：单步执行、设置断点、观察变量值。

## 2.2 实验内容

**2.2.1 源程序改错题**

下面是计算s=n!的源程序，在这个源程序中存在若干语法和逻辑错误。要求在计算机上对这个例子程序进行调试修改，使之能够正确完成指定任务。例如，8！=40320。

1 #include <stdio.h>

2 void main(void)

3 {

4 int i,n,s=1;

5 printf("Please enter n:");

6 scanf("%d",n);

7 for(i=1,i<=n,i++)

8 s=s\*i;

9 printf("%d! = %d",n,s);

10 }

解答:

1. 错误修改:
2. 第6行的变量前应加上&以取地址，正确形式为：

scanf("%d", &n ) ;

1. 第8行使用for循环语句时，循环内容应使用花括号{}括起来，正确形式为：

{s=s\*i;}

错误修改后源程序:

#include <stdio.h>

void main(void)

{

int i,n,s=1;

printf("Please enter n:");

scanf("%d",&n);

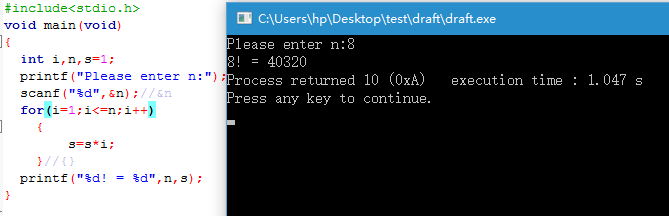
for(i=1;i<=n;i++)

{s=s\*i;}

printf("%d! = %d",n,s);

}

1. 错误修改后运行结果:



**2.2.2 源程序修改替换题**

（1）修改第1题，分别用while和do-while语句替换for语句。

解答：

1）用while语句替换for语句：

修改for循环部分的语句为：

while(n-i+1)

{

s\*=i;

i++;

}

替换后的程序如下所示：

#include<stdio.h>

void main( )

{

int i=1,n,s=1;

printf("please enter n:");

scanf("%d",&n);

while(n-i+1)

{

s\*=i;

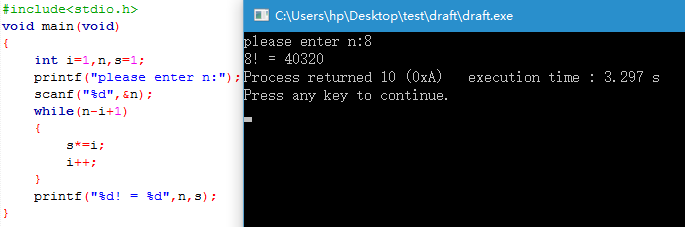
i++;

}

printf("%d! = %d",n,s);

}

运行截图：



2）用do-while语句替换for语句：

修改for循环部分的语句为：

do

{

s\*=i;

i++;

}

while(n-i+1);

替换后的程序如下所示：

#include<stdio.h>

void main( )

{

int i=1,n,s=1;

printf("please enter n:");

scanf("%d",&n);

do

{

s\*=i;

i++;

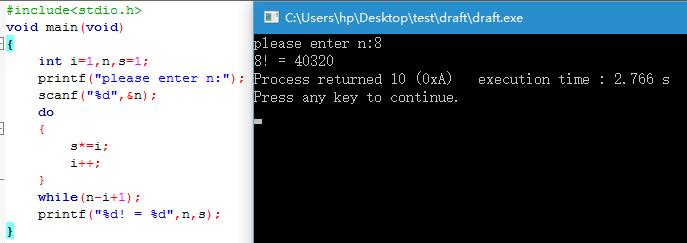
}

while(n-i+1);

printf("%d! = %d",n,s);

}

运行截图：



（2）修改第1题，输入改为“整数S”，输出改为“满足n！≥S的最小整数n”。例如输入整数40310，输出结果为n=8。

解答：

2）修改输入为整数S，即将改输入语句为：

printf("Please enter S:");

scanf("%d",&S);

在S>=2时，通过for循环用s\*=(i+1)实现阶乘，直至阶乘结果s≥S，输出阶乘中最大的因子i，即修改for循环部分的语句为：

for(i=1,s=1;s<S;i++)

{

s=s\*(i+1);

}

替换后的程序如下所示：

#include<stdio.h>

void main(void)

{

int i,S,s;

printf("Please enter S:");

scanf("%d",&S);

for(i=1,s=1;s<S;i++)

{

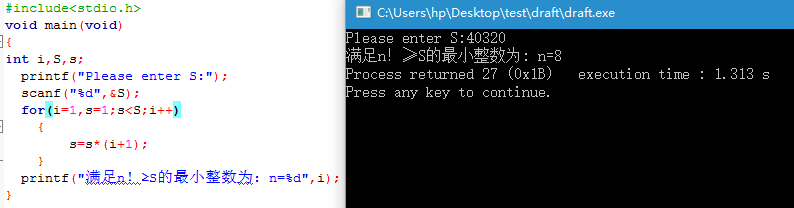
s=s\*(i+1);

}

printf("满足n！≥S的最小整数为: n=%d",i);

}

运行截图：



**2.2.3 程序设计**

（1）假设工资税金按以下方法计算：x<1000元，不收取税金；1000≤x<2000，收取5%的税金；2000≤x<3000，收取10%的税金；3000≤x<4000，收取15%的税金；4000≤x<5000，收取20%的税金；x>5000，收取25%的税金。编写一个程序，输入工资金额，输出应收取税金额度，要求分别用if语句和switch语句来实现。

解答：

1. 解题思路：
2. 提示并输入n；
3. 判断是否输入文件尾：若不是，转2.1，否则输入文件尾，转4；
   1. 判断输入n是否符合条件：如果n>0，转2.2，否则转3；
   2. 根据题目要求对不同的n用if语句实现分段处理；
   3. 通过输入工资金额n计算税金额度tax，输出结果并等待再次输入工资n，直至遇文件尾，转4。
4. 显示输入错误信息；
5. 结束。
6. 程序清单：

#include<stdio.h>

int main (void)

{

double n,tax;

while(scanf("%lf",&n)!=EOF)

{

if (n>0)

{

if(n>0&&n<=1000)

{

tax = 0;

}

else if(n>1000&&n<=2000)

{

tax = (n-1000)\*0.05;

}

else if(n>2000&&n<=3000)

{

tax = (n-2000)\*0.1 +50;

}

else if(n>3000&&n<=4000)

{

tax = (n-3000)\*0.15+150;

}

else if(n>4000&&n<=5000)

{

tax = (n-4000)\*0.2 +300;

}

else if(n>5000)

{

tax = (n-5000)\*0.25 +500;

}

printf("%lf\n",tax);

}

else

{

printf("wrong!\n");

return 0;

}

}

}

1. 测试
2. 测试数据：

为方便检测程序，选择n分别为999、4500、6000，同时尽量选取两端的字符。如表2-1所示。

表2-1 编程题2.2.3.(1)的测试数据

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| n |
| 用例1 | 999 | 计算结果 999.000000 |
| 用例2 | 4500 | 计算结果 400.000000 |
| 用例3 | 6000 | 计算结果 750.000000 |

1. 对应测试测试用例1、2、3的运行结果如图2-1所示。



图2-1 编程题2.2.3.(1)的测试用例1、2、3的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（2）编写一个程序，将输入的一行字符复制到输出，复制过程中将一个以上的空格字符用一个空格代替。

解答：

1. 算法流程如图2-2所示。

开始

输入处理数据组数

输入一组字符串并用scanf中的特殊用法%[^\n]s读取（若第一个字符为空格则用char f处理）

输出该空格并跳过之后空格直至遇到非空格

逐个字符处理字符串

当前字符是否为’\0’

当前字符

是否为空格’ ’

输出该字符

输出”\n”并等待输入下一组字符串，直至处理完所有字符串

结束

N

N

Y

Y

图2-2 编程题2.2.3.(2)的程序流程图

1. 程序清单：

#include<stdio.h>

int main()

{

int N, i;

char f,a[100];//limit a[] long 100

scanf("%d",&N);

while (N--)

{

fflush(stdin);

scanf("%c", &f);

printf("%c", f);

//getchar();

scanf(" %[^\n]s", a);

i = 0;

for (;a[i] != '\0';)

{

if (a[i] == ' ')

{

printf(" ");

while (a[i] == ' ')

{

i++;

}

}

else if (a[i] != ' ')

{

printf("%c", a[i]);

i++;

}

}

printf("\n");

}

return 0;

}

1. 测试
2. 测试数据：

为保证测试结果具有代表性，选择输入字符串分别为Hello world !、 Hello world!、 Hello world ! 。如表2-2所示。

表2-2 编程题2.2.3.(22)的测试数据

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| 字符串 |
| 用例1 | Hello world ! | Hello world ! |
| 用例2 | Hello world! | Hello world! |
| 用例3 | Hello world ! | Hello world ! |

1. 对应测试测试用例1、2、3的运行结果如图2-3所示。

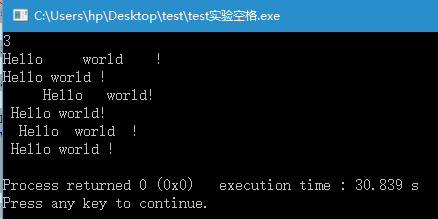


图2-3 编程题2.2.3.(2)的测试用例1、2、3的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（3）打印如下杨辉三角形。

1 /\*第0行 \*/

1 1 /\*第1行 \*/

1 2 1 /\*第2行 \*/

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

每个数据值可以由组合计算（表示第i行第j列位置的值），而的计算如下：

 (i=0,1,2,…)

 (j=0,1,2,3,…,i)

本程序中为了打印出金字塔效果，要注意空格的数目。一位数之间是3个空格，两位数之间有2个空格，3位数之间只有一个空格，程序编制过程中要注意区分。

解答：

1. 解题思路：
2. 构建函数C以实现组合函数的计算；
3. 提示输入杨辉三角行数n；
4. 判断n是否符合要求：如果1 ≤ n ≤ 12，转3.1，否则转4；
   1. 通过for循环依次对每一行进行输出；
   2. 根据计算输出当前行的第一个1以及其前面的空格，即for(space=2\*n; space>2; space--){printf(" ");}printf("1");
   3. 通过定义行数p和列数j，利用函数C计算当前行行每列的数值，并通过%4d输出以保持数字间的空格数符合要求，即for(;j<q;j++){printf("%4d",C(j,i)); }；
   4. 输出完该行后换行，进行下一行的输出，转3.2直至输出完成一组杨辉三角后；
   5. 等待输入新信息，转2，或输入文件尾，转5；
5. 显示输入错误信息并等待输入新信息，转3；
6. 结束。
7. 程序清单：

#include<stdio.h>

//C 计算组合函数

int C(int m,int n)//C 右上m 右下n

{

int i, num, I, M, N;

if(m==n)

{

return 1;

}

else

{

i=n-m;

num=i;

for(;num>1;num--)

{

i\*=(num-1);

}

I=i;//I=(n-m)!

num=m;

for(;num>1;num--)

{

m\*=(num-1);

}

M=m;//M=m!

num=n;

for(;num>1;num--)

{

n\*=(num-1);

}

N=n;//N=n!

return N/M/I;

}

}

int main()

{

int n;

int space,i,j,p;

while(scanf(" %d",&n)!=EOF)

{

if(n>=1&&n<=12)

{

p=n;//记录总行数

for(;n>0;n--)

{

for(space=2\*n; space>2; space--)

{

printf(" ");

}

printf("1");

//输出首个数字及之前空格

i=p-n;//行

j=1; //列

int q=i+1;

for(;j<q;j++)

{

printf("%4d",C(j,i));

}

printf("\n");

}

printf("\n");

}

else

{

printf("wrong!\n");

}

}

return 0;

}

1. 测试
2. 测试数据：

为使测试结果具有代表性，选择n分别为2、10、12。如表2-3所示。

表2-3 编程题2.2.3.(3)的测试数据

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| n |
| 用例1 | 2 | 符合题目要求的2阶杨辉三角 |
| 用例2 | 10 | 符合题目要求的10阶杨辉三角 |
| 用例3 | 12 | 符合题目要求的12阶杨辉三角 |

1. 对应测试测试用例1、2、3的运行结果如图2-4所示。

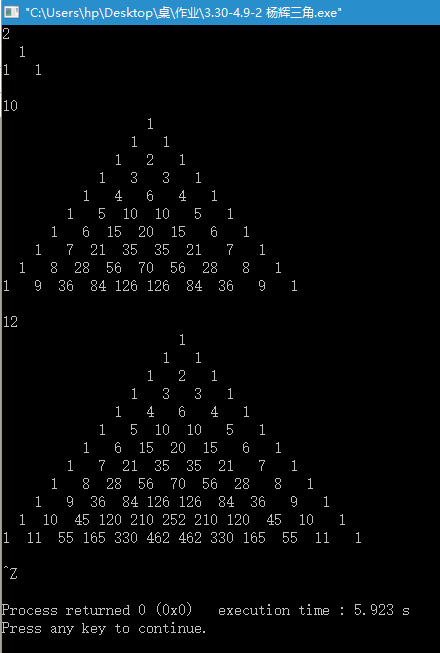


图2-4 编程题2.2.3.(3)的测试用例1、2、3的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（4）编写一个程序，将用户输入的任意正整数逆转，例如，输入1234，输出4321。

解答：

1. 解题思路：
2. 提示并输入正整数n；
3. 若n满足n>0，转2.1，否则转3：
   1. 通过for循环对正整数中的位数由低到高依次处理；
   2. 对每一位数，由高其一位的10的整数次方（如百位对应1000，千位对应10000）取模，再除以该位对应的10的整数次方（如百位对应100，千位对应1000），得到正整数中的该位数字；
   3. 每处理一低位数，将其作为新的高位数输出，处理完一个数字后输出换行符\n，并等待输入下一组数据，直至输入0（结束标志），转3；
4. 结束。
5. 程序清单：

#include<stdio.h>

int main (void)

{

int n;

while(1)

{

printf("请输入将被逆转的正整数：");

scanf("%d",&n);

if(n>0)

{

int i = 10, p=1;

for(;n/p != 0;i\*=10,p\*=10)

{

int x\_one = n%i /p;

printf ("%d", x\_one);

}

printf("\n");

}

else

return 0;

}

}

1. 测试
2. 测试数据

分别选取长度不同的整数，选择n分别为123、123456、123456789等特殊数字。如表2-4所示。

表2-4 编程题2.2.3.(4)的测试数据

|  |  |  |
| --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 |
| n |
| 用例1 | 123 | 输出结果 321 |
| 用例2 | 123456 | 输出结果 654321 |
| 用例3 | 123456789 | 输出结果 987654321 |

1. 对应测试测试用例1、2、3的运行结果如图2-5所示。

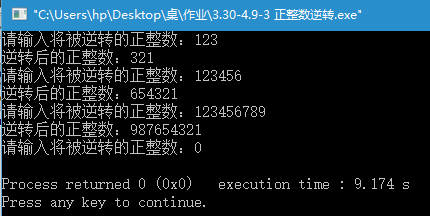


图2-5 编程题2.2.3.(4)的测试用例1、2、3的运行结果

说明上述的运行结果与理论分析吻合，验证了程序的正确性。

（5）（选做题）编写一个程序，用牛顿迭代法求方程f (x) = 3x3 - 4x2 - 5x + 10 = 0满足精度e=10-6的一个近似根，并在屏幕上输出所求近似根。

解答：

1. 解题思路：
2. 构建函数f1、f2分别实现方程f(x)的原型表达以及导函数表达；
3. 为牛顿迭代法赋一个没有限制的初值，即x0=1.0；
4. 利用函数do{x1=x0;x0=x1-f1(x1)/f2(x1);}while(fabs(x0-x1)>=0.000001)实现牛顿迭代法解方程，解得符合题目条件满足精度e=10-6的一个近似根；
5. 输出近似根，即printf("%f\n",x0);
6. 结束。
7. 程序清单：

#include<stdio.h>

#include<math.h>

float f1(float x)

{

float y;

y=x\*(x\*(3\*x-4)-5)+13;

return y;

}

float f2(float x)

{

float y;

y=x\*(9\*x-8)-5;

return y;

}

int main (void)

{

float x0,x1;

x0=1.0;

do

{

x1=x0;

x0=x1-f1(x1)/f2(x1);

}

while(fabs(x0-x1)>=0.000001);

printf("%f\n",x0);

}

1. 测试
2. 测试数据：无；
3. 测试结果：程序运行结果如图2-6。

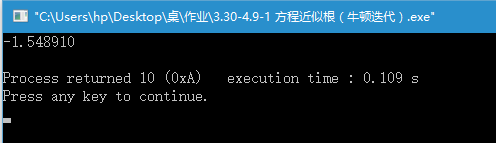


图2-6 编程题2.2.3.(5)的运行结果

程序求得的方程根是符合要求的根，说明上述的运行结果与理论分析吻合，验证了程序的正确性。

## 2.3 自设题

**（1）**自设实验题目：分别用三种循环语句（while、for、while-do）实现循环输入大于0的数字进行累加，直到输入的数字为0，结束循环，最后输出累加的结果。

**（2）**实验目的：通过设计实验程序，理解三种循环语句的相通部分以及不同点，从而能够在不同的需求下采用最合适的语句编程。

**（3）**实验程序：

while语句：

#include<stdio.h>

int main(void)

{

int N,sum=0;

while(scanf("%d",&N)!=EOF)

{

getchar();

if(N!=0)

{

sum+=N;

}

else

break;

}

printf("输入的正整数累加后结果：%d\n",sum);

}

for语句：

#include<stdio.h>

int main (void)

{

int N,sum=0;

for(;scanf("%d",&N)!=EOF;)

{

getchar();

if(N!=0)

{

sum+=N;

}

else

break;

}

printf("输入的正整数累加后结果：%d\n",sum);

}

do-while语句：

#include<stdio.h>

int main (void)

{

int N=1,sum=0;

do

{

if(N!=0)

{

sum+=N;

}

else

break;

}

while(scanf("%d",&N)!=EOF);

printf("输入的正整数累加后结果：%d\n",sum-1);

}

**（4）**实验用例：

while语句运行结果如图2-7。

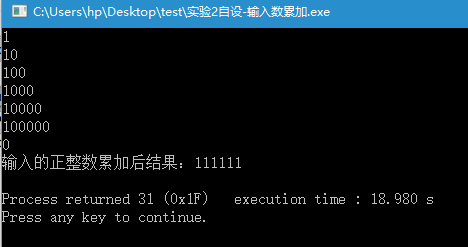


图2-7 自设题2.4 while语句的测试用例的运行结果

for语句运行结果如图2-8。

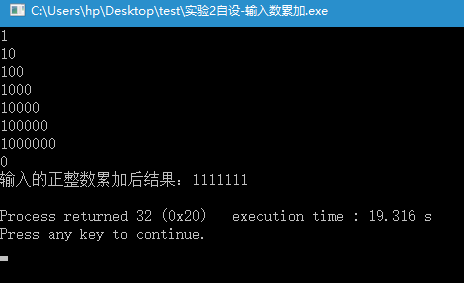


图2-8 自设题2.4 for语句的测试用例的运行结果

do-while语句运行结果如图2-9。

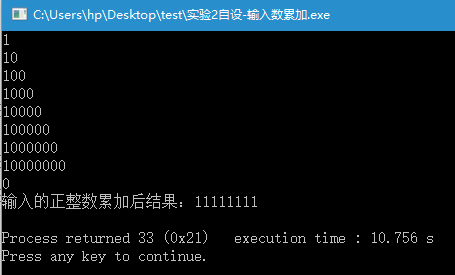


图2-9 自设题2.4 do-while语句的测试用例的运行结果

**（5）**实验结论：分别基于三种循环语句，用来实现目的的代码之间大同小异，在此程序中主要注意do-while与其他两语句的不同之处，即会先运行一次循环语句内的内容再进行判断。

## 2.4 实验小结

在实验中主要遇到的问题有：

1. 在2.2.2源程序替换的题目中为实现对程序的改动最小化，尝试了多种修改方法。改动过程中体会到了实现同一目的可使用的方式的多样性，最后也达到了自己的要求.
2. 在2.2.3程序设计的多道题目中想要使用while(scanf(“%d”,&n)!=EOF)来实现循环输入多组数据，同时遇文件尾结束。但经过尝试后发现这样的语句会使得第二次执行该语句时读入输入缓冲流中的换行符\n，故使用while(scanf(“ %d”,&n)!=EOF)来吞掉换行符\n。但这样处理过后对文件尾EOF的读取又会产生偏差，需要连续输入两次文件尾。最后经过思考在while语句内添加了getchar();语句或fflush(stdin);语句以读取输入流中不希望存在的部分，解决了问题。
3. 在解决问题2.2.3程序设计（3）杨辉三角时，构建求组合数的函数C时考虑不够周全，遗漏了一种情况下的返回值，造成程序死循环。最后通过VS2015下的调试程序跟踪变量，发现了问题所在，并成功输出了正确的杨辉三角。

体会：

1. 在编写程序时不必急于解题，应对每一种问题的方式都多种多样，就流程控制而言我们就可以使用switch、if语句以及多种循环语句（for、do-while、while）。解题前可多磨砺自己的思路，想想实现目的的方式都有哪些，如2.2.2中的程序替换就有多种方式解决同一问题，从中挑选出合适高效的算法再进行编程反而可能会简单许多；
2. 对于不同的停止程序的要求，实现停止的方式也多种多样，处理N组数据后停止、遇文件尾停止、输入0停止，每种停止方式的实现手段更是多种多样。而在尝试停止程序的同时也可能遇上多处错误（如本次2.4实验小结-遇上的问题-2中的文件尾判断），只有动手去尝试各种方法才能逐渐领会编译器的工作原理，从而找到真正正确的处理方式，这也将为我们以后可能遇到的编译大型工程打下基础；
3. 编译器报错功能并不值得太过依赖，一个程序若陷入死循环或是由于其他各式各样的原因，很容易造成死机，这时便不能依靠编译器的报错机制纠正程序。相比之下，若能够熟练掌握Turbo C 2.0集成开发环境中的调试功能：诸如单步执行、设置断点、观察变量值，便可帮助程序员更加快速而准确的找出程序可能出错的区间，提高纠错效率，从而提升编程效率。更进一步的，像断点安置地点的选择等问题，也更值得我们初学者去慢慢体会钻研。