

**课 程 实 验 报 告**

**课程名称： C语言程序设计实验**

**专业班级： 软件工程2003班**

**学 号： U202010783**

**姓 名： 刘铭宸**

**指导教师： 唐赫**

**报告日期： 2020.11.11**

**软件工程**

**目 录**

[**2 流程控制实验 1**](#_Toc404837920)

[2.1 实验目的 1](#_Toc404837921)

[2.2 实验内容 1](#_Toc404837922)

[2.3 实验小结 1](#_Toc404837923)7

# 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**

**1．程序改错**

下面的实验2-1程序是合数判断器（合数指自然数中除了能被1和本身整除外，还能被其它数整除的数），在该源程序中存在若干语法和逻辑错误。要求对该程序进行调试修改，使之能够正确完成指定任务。

/\* 实验2-1改错题程序：合数判断器\*/

#include <stdio.h>

int main( )

{

int i, x, k, flag = 0;

printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

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

for(i=2,k=x>>1;i<=k;i++)

if (!x%i) {

flag = 1;

break;

}

if(flag=1) printf("%d是合数", x);

else printf("%d不是合数", x);

}

return 0;

}

**解答：**

（1）错误修改：

1) 第7行的while循环内flag应该每次置为0，正确形式为：

flag = 0;

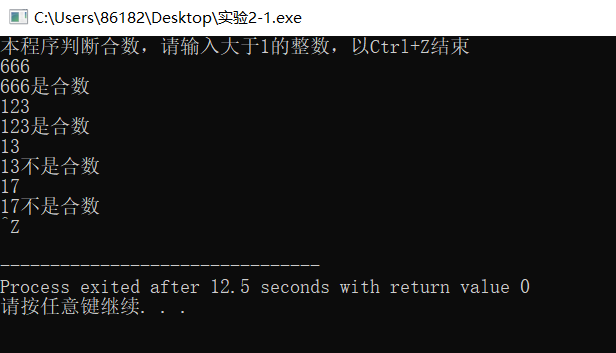
2) 第10行的!运算优先级高，产生错误运算结果，正确形式为：

if (!(x % i))

3) 第14行的判断语句应使用==，正确形式为：

if (flag == 1)

（2）错误修改后运行结果：



**2.2.2 源程序修改替换**

（1）修改实验2-1程序，将内层两出口的for循环结构改用单出口结构，即不允许使用break、goto等非结构化语句。

**解答：**

For循环中的break非结构语句破坏了单出口结构，将flag的判断放入for循环的条件控制语句，这样中途如果出现一个模数使其取模为0，即其为合数，便退出循环。

/\* 实验2-1改错题程序：合数判断器\*/

/\* 程序修改替换：改单出口 \*/

#include <stdio.h>

int main(void)

{

int i, x, k;

printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

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

int flag = 0;

for (i = 2, k = x >> 1; i <= k&&flag!=1; i++)

if (!(x % i)) flag=1;

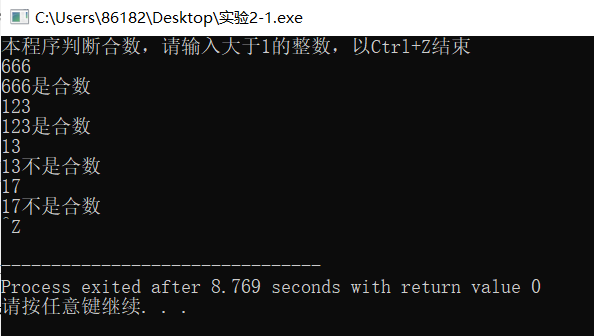
if (flag==1) printf("%d是合数\n", x);

else printf("%d不是合数\n", x);

}

return 0;

}



（2）修改实验2-1程序，将for循环改用do-while循环。

**解答：**

将for循环的判断语句放入while的判断语句，需注意do-while循环至少执行一次，所以对数字2需要特殊判断。

/\* 实验2-1改错题程序：合数判断器\*/

/\* 程序修改替换：改用dowhile循环 \*/

#include <stdio.h>

int main(void)

{

int i, x, k;

printf("本程序判断合数，请输入大于1的整数，以Ctrl+Z结束\n");

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

int flag = 0;

k=x>>1;

i=2;

do{

if(!(x%i)){

flag=1;

break;

}

i++;

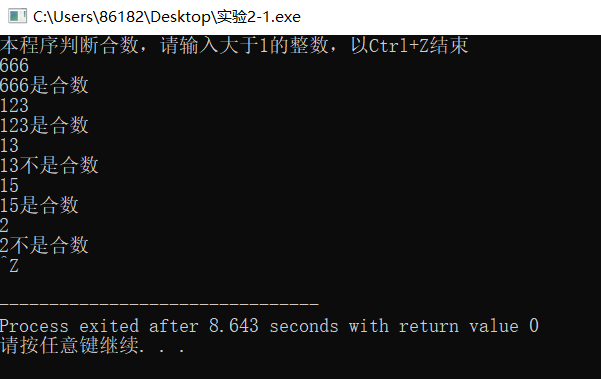
}while(i<=k);

if (flag==1 && x>2) printf("%d是合数\n", x);

else printf("%d不是合数\n", x);

}

return 0;

}

（3）修改实验2-1程序，将其改为纯粹合数求解器，求出所有的3位纯粹合数。一个合数去掉最低位，剩下的数仍是合数；再去掉剩下的数的最低位，余留下来的数还是合数，这样反复，一直到最后剩下一位数仍是合数，这样的数称为纯粹合数。

**解答：**

算法流程图如下：

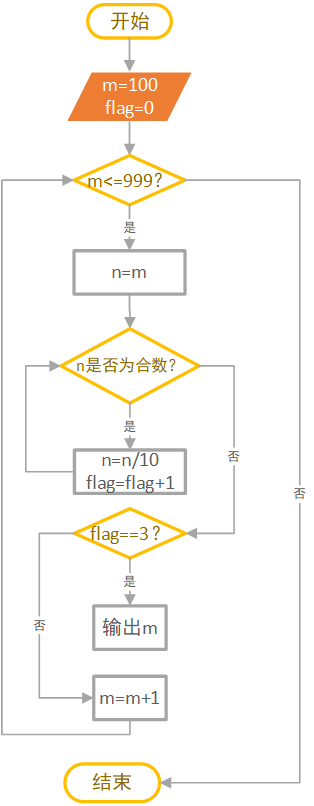


图2-1 源程序修改替换3的程序流程图

/\* 实验2-1改错题程序：合数判断器\*/

/\* 程序修改替换：纯粹合数求解器 \*/

#include <stdio.h>

int judge(int);

int main()

{

for (int m = 100; m < 1000; m++) {

int n = m, flag = 0;

while (judge(n)) {

n /= 10;

flag++;

}

if (flag == 3)printf("%d ", m);

}

return 0;

}

int judge(int x) {

int i, k, flag = 0;

for (i = 2, k = x >> 1; i <= k; i++)

if (!(x % i)) {

flag = 1;

break;

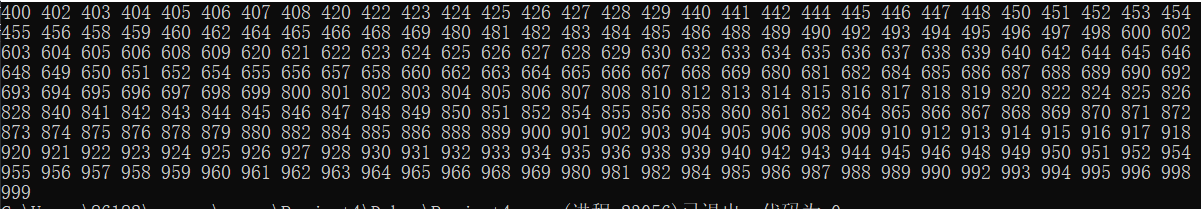
}

if (flag == 1) return 1;

else return 0;

}

程序输出展示如下：



**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；5000 ≤ x，收取25%的税金。（注意税金的计算按照阶梯计税法，比如，工资为4500，那么税金=1000\*5% + 1000\*10% + 1000\*15% + 501\*20%）。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） 本题，可使用累加的计算方法从而避免手动计算大量的公式，简单而直接。

若使用if语句：我们便注意到：

x>=5000的部分(x-4999)应该征收25%的税金，为保证阶梯计税，x=4999

x>=4000的部分(x-3999)应该征收20%的税金，为保证阶梯计税，x=3999

x>=3000的部分(x-2999)应该征收15%的税金，为保证阶梯计税，x=2999

x>=2000的部分(x-1999)应该征收10%的税金，为保证阶梯计税，x=1999

x>=1000的部分(x-999)应该征收5%的税金，为保证阶梯计税，x=999

若使用switch语句：我们注意到case必须是整数值，则可利用int(x/1000)作为参量值，使用deflaut语句避免x>=6000，的情况，我们仍然要使用累加的方法，所以我们从大到小排列case，并不break，令其一路运算到底。

2）源程序清单

#include <stdio.h>

int main() {

int x = 0; double tax = 0;

printf("请输入工资金额：");

scanf\_s("%d", &x);

if (x >= 5000) {

tax += 0.25 \* (x - 4999);

x = 4999;

}

if (x >= 4000) {

tax += 0.2 \* (x - 3999);

x = 3999;

}

if (x >= 3000) {

tax += 0.15 \* (x - 2999);

x = 2999;

}

if (x >= 2000) {

tax += 0.1 \* (x - 1999);

x = 1999;

}

if (x >= 1000) {

tax += 0.05 \* (x - 999);

x = 999;

}

printf("应收取税金为 %lf元\n", tax);

return 0;

}

#include <stdio.h>

int main() {

int x = 0; double tax = 0;

printf("请输入工资金额：");

scanf\_s("%d", &x);

switch (x / 1000) {

default:

case 5:tax += 0.25 \* (x - 4999); x = 4999;

case 4:tax += 0.2 \* (x - 3999); x = 3999;

case 3:tax += 0.15 \* (x - 2999); x = 2999;

case 2:tax += 0.1 \* (x - 1999); x = 1999;

case 1:tax += 0.05 \* (x - 999); x = 999;

case 0: break;

}

printf("应收取税金为%lf元\n", tax);

return 0;

}}3）测试

（a） 测试数据：

500

1500

2500

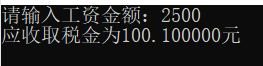
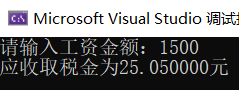
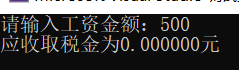
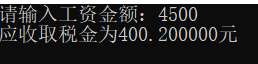
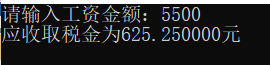
3500

4500

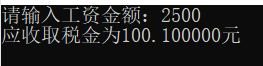
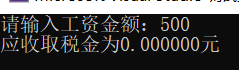
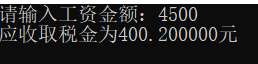
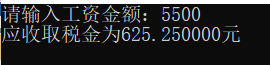
5500

（b） 对应测试数据的运行结果截图

/\*if \*/

/\* switch \*/

（2）输入一段以!结尾的短文(最多5行,每行不超过50个字符)，要求将它复制到输出，复制过程中将每行一个以上的空格字符用一个空格代替。

**解答：**

1)解题思路：

1.定义一个字符数组str用以存储读入的字符，数字i作为短文长度。

2.如果str[i-1]!=’!’，读入该字符，否则跳转3

3. 判断是否存在每行一个以上的空格，若不存在，显示输出；若存在，输出一个空格；

4. 结束

2）程序清单

#include <stdio.h>

int main(void)

{

char str[1000] = { 0 };

int i = 0, j = 0;

while (str[i - 1] != '!') {

str[i] = getchar();

i++;

}

for (j; j < i; j++) {

while (str[j] == ' ' && str[j + 1] == ' ')j++;

printf("%c", str[j]);

}

return 0;

}

3）测试

（a） 测试数据：

根据题目要求以及各种可能出现的错误情况构造数据如表1-1所示。

表2-1 编程题2的测试数据（以’#’表示空格符号）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 程 序 输 入 | 理 论 结 果 | 运 行 结 果 |
| 输入文本 |
| 用例1 | 我#爱###你  就##像#老鼠####爱  ####大米  ! | 我#爱#你  就#像#老鼠#爱  #大米  ! | 我#爱#你  就#像#老鼠#爱  #大米  ! |
| 用例2 | I#like#apples! | I#like#apples! | I#like#apples! |
| 用例3 | 生活#就像####海洋，  ####Life#is#like#a######sea. ###! | 生活#就像#海洋，  #Life#is#like#a#sea. #! | 生活#就像#海洋，  #Life#is#like#a#sea. #! |

（b） 对应测试测试用例1的运行结果如图2-2所示。

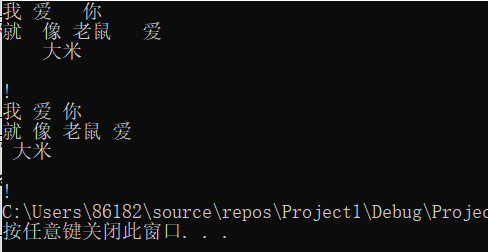


图2-2 编程题2的测试用例一的运行结果

对应测试测试用例2的运行结果如图2-3所示。



图2-3 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图2-4所示。

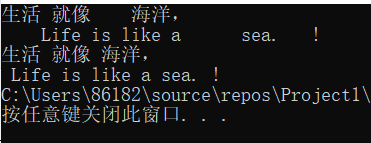


图2-4 编程题3的测试用例三的运行结果

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） 打印如下的杨辉三角形。

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

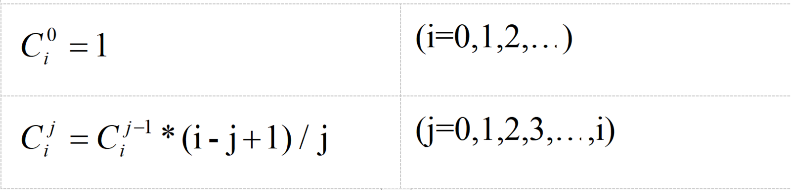
1 6 15 20 15 6 1

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

第i行第j列位置的数据值可以由组合C表示，C的计算如下：



根据以上公式，采用顺推法编程，输入最后一行的编号N（0<=N<=6），要求输出金字塔效果的杨辉三角形。

特别要注意空格的数目，每个数字占位4格，例如，1位数后补3个空格。第N行行首是N个空格（每向上一行，行首空格数对应增加）。每行末尾是换行符。

**解答：**

1) 解题思路：

1.输入N.

2.利用公式将要输出的数字存放在二维数组array中.

3.定义m表示当前行，在每行行首输出3N-2m个空格.

4.判断当前输出的数字是几位数

4.1若是一位数，则在输出该数字后输出三个空格；

4.2若是两位数，则在输出该数字后输出两个空格；

4.3若是三位数，则在输出该数字后输出一个空格；

5.结束

2）程序清单

#include <stdio.h>

int main(void) {

int N = 0;

scanf("%d", &N);

int array[100][100] = { 0 };

for (int i = 0; i < N + 1; i++) {

array[i][0] = 1;

for (int j = 1; j < i + 1; j++) {

array[i][j] = array[i][j - 1] \* (i - j + 1) / (double)j;

}

}

for (int m = 0; m < N + 1; m++) {

for (int k = 0; k < 3 \* N - 2 \* m; k++)printf(" ");

for (int n = 0; n < m + 1; n++) {

printf("%d", array[m][n]);

if (array[m][n] >= 10 && array[m][n] < 100)printf(" ");

else if (array[m][n] >= 100)printf(" ");

else printf(" ");

}

printf("\n");

}

return 0;

}

3）测试

（a） 测试数据：

根据题目要求以及各种可能出现的错误情况构造数据如表1-2所示。

表2-2 编程题3的测试数据

|  |  |  |  |
| --- | --- | --- | --- |
| 测试  用例 | 输 入 | 理 论 结 果 | 运 行 结 果 |
| N |
| 例1 | 0 | 1 | 1 |
| 例2 | 3 | 1  1 1  1 2 1  1 3 3 1 | 1  1 1  1 2 1  1 3 3 1 |
| 例3 | 6 | 1  1 1  1 2 1  1 3 3 1  1 4 6 4 1  1 5 10 10 5 1  1 6 15 20 15 6 1 | 1  1 1  1 2 1  1 3 3 1  1 4 6 4 1  1 5 10 10 5 1  1 6 15 20 15 6 1 |

（b） 对应测试测试用例1的运行结果如图2-5所示。



图2-5 编程题3的测试用例一的运行结果

对应测试测试用例2的运行结果如图2-6所示。

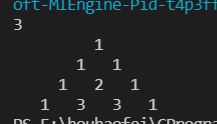


图2-6 编程题3的测试用例二的运行结果

对应测试测试用例3的运行结果如图2-7所示。

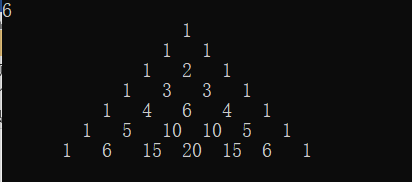


图2-7 编程题3的测试用例三的运行结果

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） 625这个数很特别，625的平方等于390625，其末3位也是625。请编程输出所有这样的3位数：它的平方的末3位是这个数本身。要求这些数字从小到大排列，每个数字单独占一行。

**解答：**

1. 解题思路：算法流程可用算法流程图表示

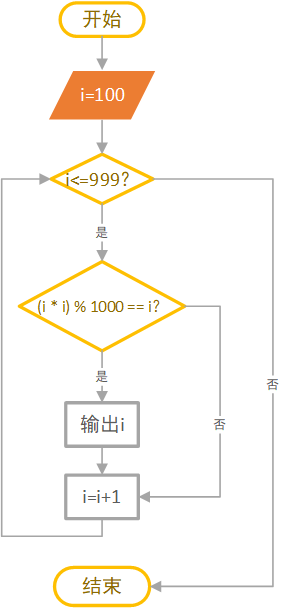


图2-8 编程题4的程序流程图

2）程序清单

#include <stdio.h>

int main(void) {

for (int i = 100; i < 1000; i++)

if ((i \* i) % 1000 == i)printf("%d\n", i);

return 0;

}

3）测试

（a） 测试数据：

由数学推导已知，这类数的个数是有限的，可演算得仅2个三位数满足条件，即376和625。

（b） 对应测试的运行结果如图2-8所示。

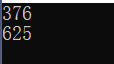


图2-9 编程题4的测试的运行结果

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。

## 2.3 实验小结

主要叙述实验过程中遇到的问题，如何解决的，通过分析、结果问题后的体会。

在实验1源程序改错中，采用逐步调试的方法成功找出了语法上的错误，最终解决了所有改错。

在修改题1中不知道如何修改实验2-1程序，将内层两出口的for循环结构改用单出口结构，即不允许使用break、goto等非结构化语句。和同学交流过后，明白了单出口的含义，For循环中的break非结构语句破坏了单出口结构，将flag的判断放入for循环的条件控制语句，这样中途如果出现一个模数使其取模为0，即其为合数，便退出循环。既没有破坏单出口的性质，又保证了运算的效率。

在编程题3中，学会了仔细阅读题面然后逐渐分析得出空格的个数解决了问题，明白了学会审题的重要意义。