

# 云南大学数学与统计学实验教学中心

## 《高级语言程序设计》实验报告

课程名称：程序设计和算法语言	学期：2016~2017 学年上学期	成绩：
指导教师：赵越	学生姓名：刘鹏	学生学号：20151910042
实验名称：模块化程序设计		
实验编号：No.06	实验日期：2018 年 8 月 19 日	实验学时：2
学院：数学与统计学院	专业：信息与计算科学	年级：2015 级

### 一、实验目的

掌握 C 语言中定义函数的方法及其调用方法。

掌握函数实参与形参的对应关系以及“值传递”与“地址传递”的方式。

掌握全局变量和局部变量、动态变量与静态变量的概念和使用方法。

掌握函数的嵌套调用与递归调用的方法。

掌握通过数组进行数据传递的方法。

学会使用宏替换编写程序，弄清“文件包含”的作用。

学会全局变量和局部变量、动态变量和静态变量的概念和使用方法。

### 二、实验环境

Windows10 Pro Workstation 17134.288;

Cygwin GCC 编译器。

### 三、实验内容

1. 变量的生存期是一个静态的概念，是不变的，作用域是一个动态的概念，随程序的运行而变。从作用域角度分，有**局部变量**和**全局变量**；从变量存在的时间来分，有**动态存贮**和**静态存贮**。

2. C 语言中的参数调用为值调用，名调用（地址）通过指针实现的。

3. 用递归方式编写程序，一般由三部分组成：

递归条件；

递归化简；

递归出口；

一个递归总是可分为“回推”和“递推”两个阶段，即从要解决问题出发，一步步推到已知条件，然后再由已知条件一步步找到结论。

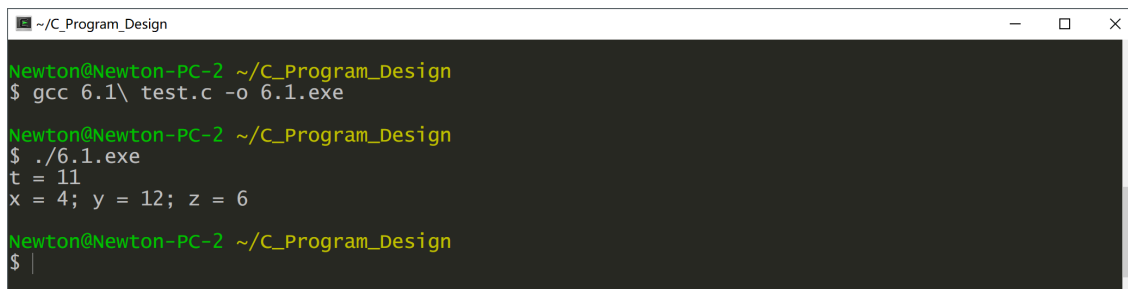
#### 3.1 函数调用练习

通过运行下面程序，熟悉函数的调用方法。

### 3.1.1 程序代码

```
1  /*
2  * filename: 6.1 test.c
3  * property: test
4  */
5
6  #include <stdio.h>
7
8  void fun(int i, int j, int k) {
9      int t;
10     t=(i + j + k) / 2;
11     printf("t = %d\n", t);
12 }
13
14 int main() {
15     int x, y, z;
16     x = 4;
17     y = 12;
18     z = 6;
19     fun(x, y, z);
20     printf("x = %d; y = %d; z = %d\n", x, y, z);
21     return 0;
22 }
```

### 3.1.2 运行结果



```
~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.1\ test.c -o 6.1.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.1.exe
t = 11
x = 4; y = 12; z = 6

Newton@Newton-PC-2 ~/C_Program_Design
$ |
```

## 3.2 程序调用

运行下面程序，写出执行结果。

### 3.2.1 程序代码

```
1  /*
2  * filename: 6.2 test.c
3  * property: test
4  */
5
6  #include <stdio.h>
7
8  void f(int i,int j) {
9      int x, y, g;
10     g = 8;
```

```

11     x = 7;
12     y = 2;
13     printf("g = %d; i = %d; j = %d\n", g, i, j);
14     printf("x = %d; y = %d\n", x, y);
15 }
16
17 int main() {
18     int i, j, x, y, n, g;
19     i = 4;
20     j = 5;
21     g = x = 6;
22     y = 9;
23     n = 7;
24     f(n, 6);
25     printf("g = %d; i = %d; j = %d\n", g, i, j);
26     printf("x = %d; y = %d\n", x, y);
27     f(n, 8);
28     return 0;
29 }

```

### 3.2.2 运行结果

```

~/_C_Program_Design
Newton@Newton-PC-2 ~/_C_Program_Design
$ gcc 6.2\ test.c -o 6.2.exe

Newton@Newton-PC-2 ~/_C_Program_Design
$ ./6.2.exe
g = 8; i = 7; j = 6
x = 7; y = 2
g = 6; i = 4; j = 5
x = 6; y = 9
g = 8; i = 7; j = 8
x = 7; y = 2

Newton@Newton-PC-2 ~/_C_Program_Design
$

```

## 3.3 素数判断

编写一个判断素数的函数，在主函数输入一个整数，输出是否是素数的信息。

### 3.3.1 程序代码

```

1  /*
2  * filename: 6.3 prime.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7  #include <math.h>
8
9  void Prime(int n) {
10     int i, k;
11     k = n/2;
12     for(i = 2; i <= k; i++)
13         if (n % i == 0)

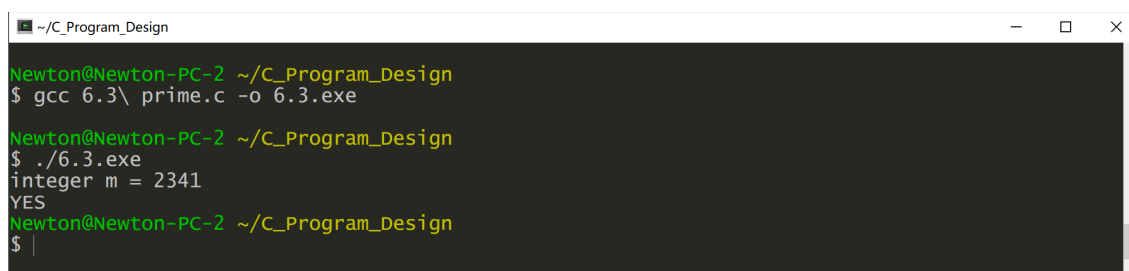
```

```

14         break;
15
16     if(i >= k + 1) {
17         printf("YES");
18     }
19     else {
20         printf("NO");
21     }
22 }
23
24 int main() {
25     int m;
26     printf("integer m = ");
27     scanf("%d",&m);
28     Prime(m);
29     return 0;
30 }

```

### 3.3.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.3\ prime.c -o 6.3.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.3.exe
integer m = 2341
YES
Newton@Newton-PC-2 ~/C_Program_Design
$ |

```

## 3.4 代码分析

先读懂程序，分析出结果，然后上机运行此程序。

### 3.4.1 程序代码

```

1  /*
2  * filename: 6.4 analysis.c
3  * property: analysis
4  */
5
6  #include <stdio.h>
7
8  #define FUE(K)          K + 3.14159
9  #define PR(a)           printf("a = %d\t", (int)(a))
10 #define PRINT(a)         PR(a); putchar('\n')
11 #define PRINT2(a, b)     PR(a); PRINT(b)
12 #define PRINT3(a, b, c) PR(a); PRINT2(b,c)
13 #define MAX(a, b)        (a < b? b : a)
14
15 int main() {
16     {
17         int x = 2;

```

```

18     PRINT(x * FUE(4));
19 }
20
21 {
22     int f;
23     for(f = 0; f <= 60; f += 20) {
24         PRINT2(f, 5.12 * f + 45);
25     }
26 }
27
28 {
29     int x = 1, y = 2;
30     PRINT3(MAX(x++, y), x, y);
31     PRINT3(MAX(x++, y), x, y);
32 }
33     return 0;
34 }

```

### 3.4.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.4\ analysis.c -o 6.4.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.4.exe
a = 11
a = 0    a = 45
a = 20   a = 147
a = 40   a = 249
a = 60   a = 352
a = 2    a = 2    a = 2
a = 3    a = 4    a = 2
Newton@Newton-PC-2 ~/C_Program_Design
$

```

可以看出，当进行了宏定义之后，不加括号与加括号是有很大的区别的。

## 3.5 \*统计字符串的字符信息

编写一函数，由实参传来一个字符串，统计此字符串中字母、数字、空格和其它字符的个数，在主函数中输入字符串，输出上述结果。此程序编写时最好把存放字母、数字、空格和其它字符的变量定义为全程变量，这样在函数中就不必定义了。用其它方法也可以。

### 3.5.1 程序代码

```

1  /*
2  * filename: 6.5 string info.c
3  * property: exercise
4  */
5
6  #include <string.h>
7  #include <stdio.h>
8
9  int main() {

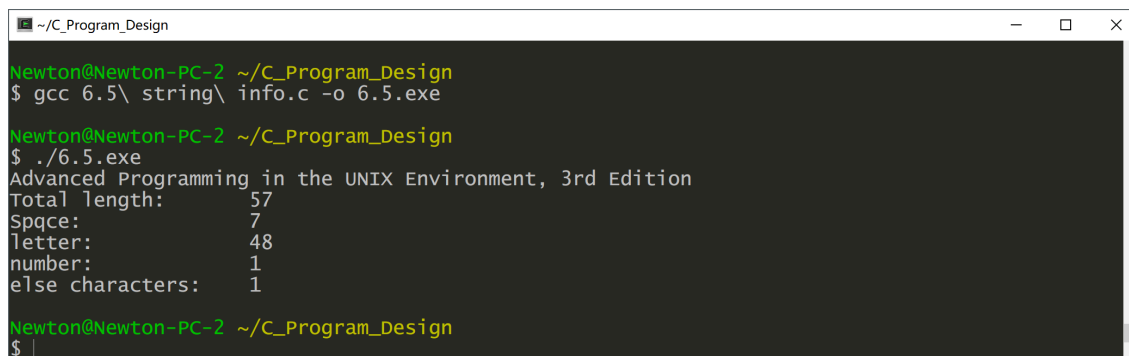
```

```

10  int space = 0;
11  int alpha = 0;
12  int numbs = 0;
13  int elsec = 0;
14
15  char a[1000];
16  gets(a);
17
18  int i;
19
20  for (i = 0; i <= strlen(a); i++) {
21      if (a[i] >= 'a' && a[i] <= 'z' || a[i] >= 'A' && a[i] <= 'Z') {
22          alpha += 1;
23      }
24      if (a[i] == ' ') {
25          space += 1;
26      }
27      if (a[i] >= '0' && a[i] <= '9') {
28          numbs += 1;
29      }
30  }
31
32  elsec = strlen(a) - space - alpha - numbs;
33  printf("Total length:      %d\n", strlen(a));
34  printf("Spqce:            %d\n", space);
35  printf("letter:            %d\n", alpha);
36  printf("number:            %d\n", numbs);
37  printf("else characters:    %d\n", elsec);
38
39  return 0;
40 }

```

### 3.5.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.5\ string\ info.c -o 6.5.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.5.exe
Advanced Programming in the UNIX Environment, 3rd Edition
Total length:      57
Spqce:            7
letter:            48
number:            1
else characters:    1

Newton@Newton-PC-2 ~/C_Program_Design
$

```

### 3.6 递归函数编写

写出计算 Ackermann 函数  $Ack(m, n)$  的递归函数, 对于  $m \geq 0, n \geq 0$ ,  $Ack(m, n)$  定义为:

$$\begin{aligned}
 Ack(0, n) &= n + 1 \\
 Ack(m, 0) &= Ack(m - 1, 1)
 \end{aligned}$$

$$\text{Ack}(m, n) = \text{Ack}((m - 1), \text{Ack}(m, n - 1))$$

对于  $m > 0, n > 0$ , 要显示出计算结果。编程并上机调试。

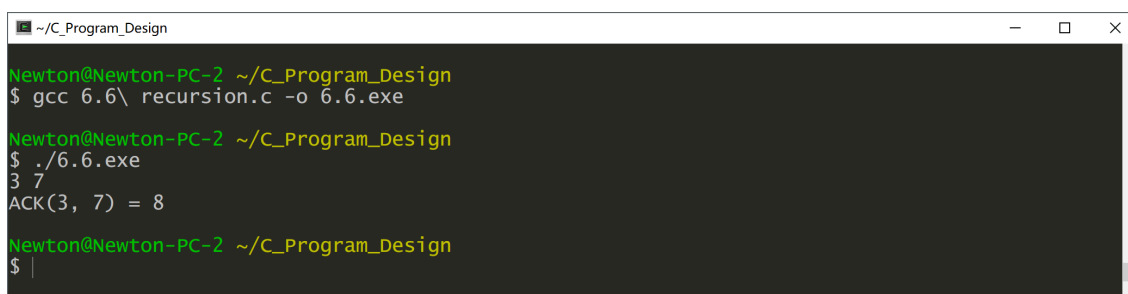
### 3.6.1 程序代码

```

1  /*
2  * filename: 6.6 recursion.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  int ACK(int m, int n) {
9      if(m == 0 && n >= 0)
10         return 1 + n;
11     else
12         if(n == 0 && m > 0)
13             return ACK(m - 1, 0);
14         else
15             if(m > 0 && n > 0)
16                 return ACK(m - 1, ACK(m, n - 1));
17 }
18
19 int main() {
20     int m, n;
21     scanf("%d %d", &m, &n);
22     printf("ACK(%d, %d) = %d\n", m, n, ACK(m, n));
23     getchar();
24 }

```

### 3.6.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/_C_Program_Design
$ gcc 6.6\ recursion.c -o 6.6.exe
Newton@Newton-PC-2 ~/_C_Program_Design
$ ./6.6.exe
3 7
ACK(3, 7) = 8
Newton@Newton-PC-2 ~/_C_Program_Design
$ |

```

## 3.7 \*最长单词

写一函数，输入一行字符，将此字符串中最长的单词输出。编写程序并上机调试运行，记录下程序运行的结果。提示：我们认为每行字符都是由字母组成的字符串，在程序中设 *longest()* 函数为实现寻找最长的字符串。设标记 *flag* 表示单词是否开始，*flag=1* 表示单词开始，*flag=0* 表示单词未开始。*Point* 代表当前单词的起始位置，*place* 代表最长单词的起始位置。*Len* 代表当前单词已累计字母的个数，*length* 代表先前单词中最长单词的长度。下面给一个程序段，上实现寻找最长的单词。

### 3.7.1 程序代码

```
1  /*
2  * filename: 6.7 longest word.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7  #include <string.h>
8
9  int is_letter(char c) {
10     if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z') {
11         return 1;
12     }
13     else {
14         return 0;
15     }
16 }
17
18 int main() {
19     char a[5000];
20     gets(a);
21
22     int max = 0;
23     int max_position = 0;
24     int word_length = 0;
25
26     int i;
27     for (i = 0; i <= strlen(a); i++) {
28         if (is_letter(a[i]) == 1) {
29             word_length += 1;
30         }
31         else {
32             if (word_length > max) {
33                 max = word_length;
34                 max_position = i - max;
35             }
36             word_length = 0;
37         }
38     }
39
40
41     for (i = max_position; i < max_position + max; i++) {
42         printf("%c", a[i]);
43     }
44     return 0;
45 }
```



### 3.7.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.7\ longest\ word.c -o 6.7.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.7.exe
Advanced Programming in the UNIX Environment
Programming
Newton@Newton-PC-2 ~/C_Program_Design
$

```

### 3.8 最大公因数与最小公倍数

求两整数的最大公约数和最小公倍数。用一函数求最大公约数，用另一函数调用此函数求出最大公约数，并用求出的最大公约数求最小公倍数。

具体要求如下：

- ①用全局变量。将最大公约数与最小公倍数设为全局变量，在主函数中输出它们的值。
- ②不用全局变量。最大公约数和最小公倍数由被调模块返回值。

#### 3.8.1 程序代码

```

1  /*
2  * filename: 6.8 gcd.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  int gcd(int a, int b) {
9      if (a == 0 || b == 0) {
10         return a + b;
11     }
12     else {
13         if (a > b) {
14             return gcd(a - b, b);
15         }
16         else {
17             return gcd(a, b - a);
18         }
19     }
20 }
21
22 int main() {
23     int a, b;
24     printf("a ,b = ");
25     scanf("%d, %d", &a, &b);
26
27     printf("gcd(a, b) = %d\n", gcd(a, b));
28     printf("lcm(a, b) = %d\n", a * b / gcd(a, b));
29

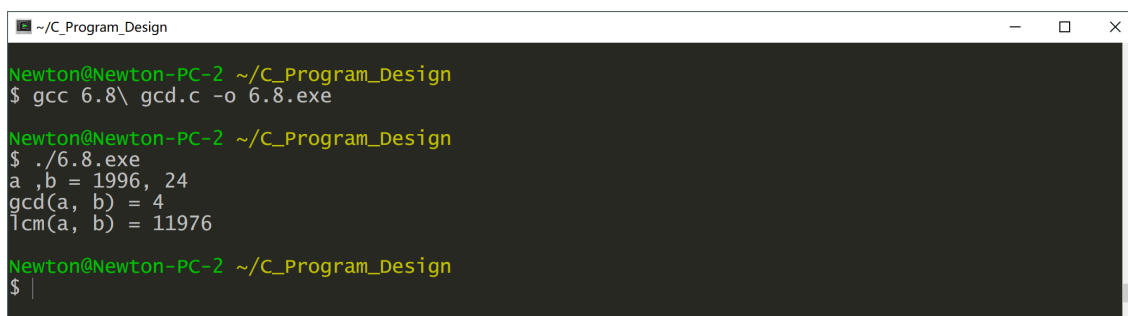
```

```

30     return 0;
31 }

```

### 3.8.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.8\ gcd.c -o 6.8.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.8.exe
a ,b = 1996, 24
gcd(a, b) = 4
lcm(a, b) = 11976
Newton@Newton-PC-2 ~/C_Program_Design
$

```

## 3.9 组合数计算

计算并输出

$$\frac{m!}{(m-n)!n!}$$

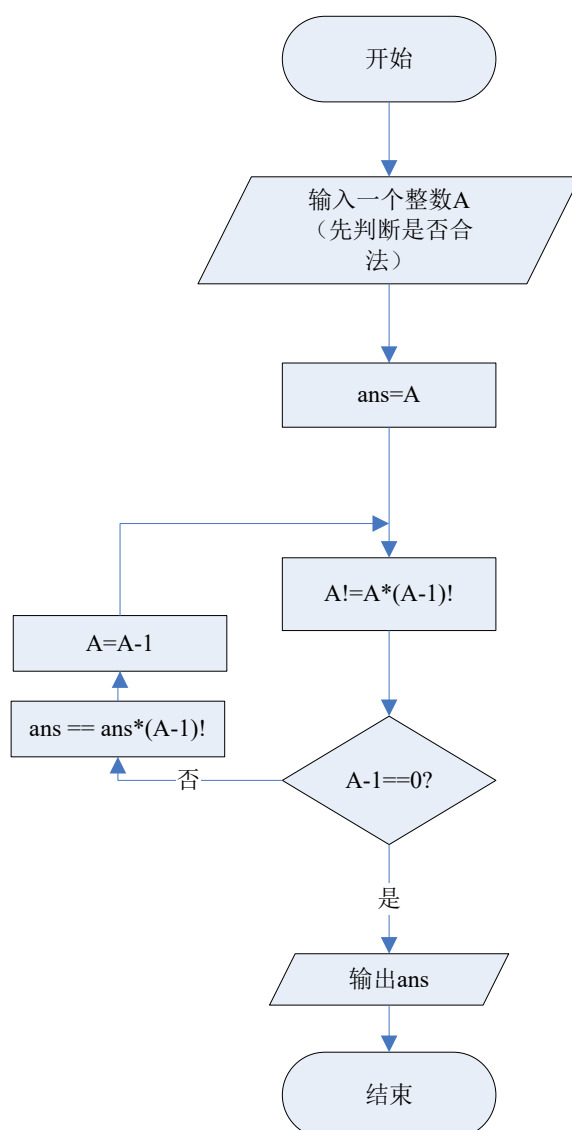
具体要求如下：

- ①编制一个函数**pq**(*n*)，返回*n*!值。
- ②编制主函数，由键盘输入*m*与*n*( $m \geq n \geq 0$ )，调用**pq**(*n*)计算下列算式值。

$$\frac{m!}{(m-n)!n!}$$

- ③在主函数中，输入*m*与*n*之前要有提示，并检查输入数据的合理性，对于不合理的输入，应输出出错信息，不再进行计算。在函数**pq**(*n*)中也要检查*n*的合理性，当*n* < 0时输出出错信息，不再进行计算。
- ④分别输入(*m*, *n*) = (3, -1), (0, 0), (8, 3), (3, 8), (8, 8)运行该程序。
- ⑤画出模块**pq**()的流程图。

## 3.9.1 流程图



## 3.9.2 程序代码

```

1  /*
2  * filename: 6.9 combinatorial number.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  #define INPUT_ERROR    -1
9
10 int factorial(int a) {
11     if (a < 0) {
12         return INPUT_ERROR;
13     }
14     if (a == 0) {
15         return 1;

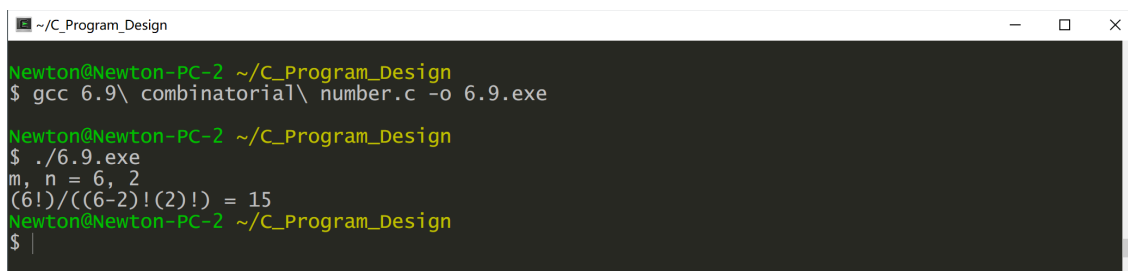
```

```

16     }
17     if (a > 1) {
18         return a * factorial(a - 1);
19     }
20 }
21
22 int main() {
23     int m, n;
24     printf("m, n = ");
25     scanf("%d, %d", &m, &n);
26
27     if (m * n <= 0 || m < n) {
28         return INPUT_ERROR;
29     }
30     else {
31         int t = factorial(m) / (factorial(n) * factorial(m - n));
32         printf("(%d!)/((%d-%d)!(%d)!) = %d", m, m, n, n, t);
33     }
34     return 0;
35 }

```

### 3.9.3 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.9\combinatorial\ number.c -o 6.9.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.9.exe
m, n = 6, 2
(6!)/((6-2)!(2)!) = 15
Newton@Newton-PC-2 ~/C_Program_Design
$

```

## 3.10 判别

编写程序，要求找出满足下列条件的 3 位数：它是完全平方数，又有两位数字相同。如：144、676。

要求：设计一函数判断一个三位数是否为完全平方数，设计另一函数判断一个三位数中是否有两位数字相同，再在主函数中调用这两个函数，找出所有的满足这两个条件的三位数。

### 3.10.1 程序代码

```

1  /*
2  * filename: 6.10 sieve.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7  #include <math.h>
8
9  int judge(int a, int b, int c) {
10     int BOOL;
11     float z;

```

```

12     float k;
13     float j;
14     z = 100 * a + 10 * b + c;
15     k = sqrt(z);
16     for(j = 1; j < z; j++) {
17         if(j - k == 0)
18             BOOL = 1;
19         continue;
20     }
21     return BOOL;
22 }
23
24 int compare(int a, int b, int c) {
25     int BOOL;
26     if((a == b && a != c) || (a == c && a != b) || (b == c && a != b)) {
27         BOOL = 1;
28     }
29     return BOOL;
30 }
31
32 int main() {
33     int a, b, c;
34     printf("a, b, c = ");
35     scanf("%d, %d, %d", &a, &b, &c);
36     if(judge(a, b, c) == 1 && compare(a, b, c) == 1) {
37         printf("%d%d%d, yes\n", a, b, c);
38     }
39     else{
40         printf("Sorry, it isn't.");
41     }
42 }

```

### 3.10.2 运行结果

```

~/_C_Program_Design
Newton@Newton-PC-2 ~/_C_Program_Design
$ gcc 6.10\ sieve.c -o 6.10.exe

Newton@Newton-PC-2 ~/_C_Program_Design
$ ./6.10.exe
a, b, c = 1, 4, 4
144, yes

Newton@Newton-PC-2 ~/_C_Program_Design
$

```

### 3.11

编写函数，将 $n$ 个整数的数列进行重新排放，重新排放后的结果为：前段都是奇数，后段都是偶数，并编写主函数完成：

- ①输入 10 个整数；
- ②调用此函数进行重排；
- ③输出重排后的结果。

### 3.11.1 程序代码

```
1  /*
2  * filename: 6.11 rearrange.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  void arrange(int a[]) {
9      int count = 0;
10     int i;
11     int ans[10];
12
13     for (i = 0; i < 10; i++) {
14         if (a[i] % 2 == 1) {
15             ans[count] = a[i];
16             count += 1;
17         }
18     }
19     for (i = 0; i < 10; i++) {
20         if (a[i] % 2 == 0) {
21             ans[count] = a[i];
22             count += 1;
23         }
24     }
25
26     for (i = 0; i < 10; i++) {
27         a[i] = ans[i];
28     }
29 }
30
31 int main() {
32     int i;
33     int a[10];
34     printf("a[10] = ");
35     for (i = 0; i < 10; i++) {
36         scanf("%d", &a[i]);
37     }
38
39     arrange(a);
40
41     for (i = 0; i < 10; i++) {
42         printf("%4d", a[i]);
43     }
44
45     return 0;
46 }
```

### 3.11.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.11\ rearrange.c -o 6.11.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.11.exe
a[10] = 1 2 3 4 5 6 7 8 9 10
      1 3 5 7 9 2 4 6 8 10
Newton@Newton-PC-2 ~/C_Program_Design
$ |

```

### 3.12 学生成绩统计

输入 10 个学生 4 门课的成绩，分别用函数求：

- ①每个学生的平均成绩；
- ②每门课的及格率；
- ③最高分所对应的学生和课程。

#### 3.12.1 程序代码

```

1  /*
2  * filename: 6.12 grade.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  #define ROW      3
9  #define COLUMN   4
10
11 void mean(int a[ROW][COLUMN]) {
12     int sum[ROW];
13     int avg[ROW];
14     int i, j;
15     for(j = 0; j < ROW; j++) {
16         sum[j] = 0;
17         for(i = 0; i < COLUMN; i++) {
18             sum[j] += a[j][i];
19         }
20         avg[j] = sum[j] / COLUMN;
21     }
22     for(j = 0; j < ROW; j++) {
23         printf("%4d\n", avg[j]);
24     }
25     printf("\n");
26 }
27
28 void passrate(int a[ROW][COLUMN]) {
29     float count[COLUMN];
30     int i, j;
31     for(j = 0; j < COLUMN; j++) {

```

```

32     for(i = 0; i < ROW; i++) {
33         if(a[i][j] >= 60) {
34             count[j]++;
35         }
36     }
37 }
38 for(i = 0; i < COLUMN; i++) {
39     printf("Course %d passrage: %.2f\n", i + 1, (count[i] / ROW) * 100.0);
40 }
41 }
42
43 int max2(int a,int b) {
44     return(a > b? a : b);
45 }
46
47 void max(int a[ROW][COLUMN]) {
48     int Max;
49     Max = a[0][0];
50     int column, row;
51     int i, j;
52     for(i = 0; i < ROW; i++) {
53         for(j = 0; j < COLUMN; j++) {
54             if(Max < a[i][j]) {
55                 Max = a[i][j];
56                 row = i;
57                 column = j;
58             }
59         }
60     }
61     printf("Student %d in Course %d, highest score = %d.\n", row + 1, column + 1, Max);
62 }
63
64 int main() {
65     int i, j;
66     int a[ROW][COLUMN];
67     printf("input score table\n");
68     for (i = 0; i < ROW; i++) {
69         for (j = 0; j < COLUMN; j++) {
70             scanf("%d", &a[i][j]);
71         }
72     }
73     printf("\nscore table:\n");
74     char space[] = "          ";
75     printf("%s", space);
76     for (i = 1; i <= COLUMN; i++) {
77         printf("course %d  ", i);
78     }
79     printf("\n");
80

```



```

81     for (i = 0; i < 8 * COLUMN + 11 + 3 * (COLUMN - 1); i++) {
82         printf("-");
83     }
84     printf("\n");
85
86     for (i = 0; i < ROW; i++) {
87         printf("Student %d", i + 1);
88         for(j = 0; j < COLUMN; j++) {
89             printf("%d", a[i][j]);
90             if(j == COLUMN-1) {
91                 printf("\n");
92             }
93         }
94     }
95     printf("\nThe mean_values of each student are:\n");
96     mean(a);
97     printf("The pass_rate of each class are:\n");
98     passrate(a);
99     printf("The highest score owner's location and score is:\n");
100    max(a);
101    return 0;
102 }

```

### 3.12.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.12\ grade.c -o 6.12.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.12.exe
input score table
68 78 52 49
99 89 96 44
71 78 79 32

score table:
      course 1   course 2   course 3   course 4
-----
Student 1      68        78        52        49
Student 2      99        89        96        44
Student 3      71        78        79        32

The mean_values of each student are:
61
82
65

The pass_rate of each class are:
Course 1 passrage: 100.00
Course 2 passrage: 100.00
Course 3 passrage: 66.67
Course 4 passrage: 0.00
The highest score owner's location and score is:
Student 2 in Course 1, highest score = 99.

Newton@Newton-PC-2 ~/C_Program_Design
$

```

### 3.13 习题

1. 下列说法哪些是不正确的

1) C 语言的函数必须在主函数的前面定义。(×)

- 2) 函数名前面都必须写数据类型和存储类型。(×)
- 3) 一个 C 程序可以包含多个函数，并且必须有最多有一个主函数。(√)
- 4) 函数的外部不允许再说明其它变量。(×)
- 5) void 类函数不能有返回值。(√)
- 6) 函数不一定都有返回值，不管其类型是否为 void。(×)
- 7) 引用程序中的函数被调用函数必须在调用函数前面预先定义。(√)
- 8) 在函数内部定义的变量都是局部量。(√)
- 9) 所有外部量，都必须在其它程序中定义过。
- 10) 函数返回的数据类型必须和函数数据类型一致。(√)
- 11) 调用函数的实参与被调用函数的形参必须保证其个数、次序类型完全一致。(√)
- 12) 在 C 程序中，函数嵌套调用不允许超过 18 层。(×)

### 3.14 程序排错

下列定义有何错误？

```

1  /*-----*/
2  float f(x,y);
3  int x,y;
4  {
5
6  }
7  /*-----*/
8  f(int x,y)
9  {
10     retuen x*y;
11 }
12 /*-----*/
13 /*-----*/
14 int max(x,y) /*定义求最大值的函数*/
15 { int x,y,z;
16     z=x>y? x:y;
17     return z;
18 }
19 (4)
20 int f(x,y,z)
21 int x,y;
22 {
23     int z;
24     :
25 }
26 /*-----*/
27 int f(int n) /*定义求 n!的函数*/
28 {
29     if n<=0 return 1;
30     else f=n*f(n-1);
31 }
32 /*-----*/

```

```

33 int main(x,y)
34 int x,y;
35 {
36     printf("x+y=%d\n",x+y);
37 }
38 int x,y;
39 f(int x)
40 {
41     y=x*x; return y ;}
42 int f1(x)
43 int x;
44 {
45     int f2(y)
46     int y;
47 { ...
48
49 }
50 }

```

### 3.15 多态函数

定义一个函数，调用程序通过  $f(n,x)$  的形式就可能计算  $x^3 + x - 1$ ,  $(5 + x)^3 + (5 + x) - 1$ ,  $(\sin x)^3 + \sin x - 1$  等样式的表达式的值。

#### 3.15.1 程序代码

```

1  /*
2  * filename: 6.13 function.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7  #include <math.h>
8
9  void f(int n, double x) {
10     switch(n) {
11         case 1: printf("%f", pow(x, 3) + x - 1); break;
12         case 2: printf("%f", pow((x + 5), 3) + (x + 5) - 1); break;
13         case 3: printf("%f", pow(sin(x), 3) + sin(x) - 1); break;
14         default: printf("Error input");
15     }
16 }
17
18 int main() {
19     printf("n, x = ");
20     int n;
21     double x;
22     scanf("%d, %f", &n, &x);
23     printf("f(n, x) = ");
24     f(n, x);

```

```

25     return 0;
26 }

```

### 3.15.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.13\ funciton.c -o 6.13.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.13.exe
n, x = 2, 3.2
f(n, x) = 129.000000
Newton@Newton-PC-2 ~/C_Program_Design
$

```

### 3.16 递归函数

写一递归函数，计算

$$c(m, n) = \begin{cases} 1, & n = 0 \text{ or } m = n \\ m, & n = 1 \\ c(m-1, n-1) + c(n, m-1), & m > n > 1 \end{cases}$$

#### 3.16.1 程序代码

```

1  /*
2  * filename: 6.14 recursion.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  int c(int m, int n) {
9      if (n == 0 || m == n) {
10         return 1;
11     }
12     if (n == 1) {
13         return m;
14     }
15     if (m > n > 1) {
16         return c(m - 1, n - 1) + c(n, m - 1);
17     }
18 }
19
20 int main() {
21     int m, n;
22     printf("m, n = ");
23     scanf("%d, %d", &m, &n);
24
25     printf("c(m, n) = %d", c(m, n));
26 }

```

### 3.16.2 运行结果

```
~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.14\ recursion.c -o 6.14.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.14.exe
m, n = 1, 1
c(m, n) = 1
Newton@Newton-PC-2 ~/C_Program_Design
$ |
```

### 3.17 反向输出

写一递归函数，将读入的整数按位分开后以相反顺序输出。

#### 3.17.1 程序代码

```
1  /*
2  * filename: 6.15 reverse output.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  int f(int n) {
9      int z;
10     if (n < 10) {
11         printf("%d", n);
12         return 0;
13     }
14     else {
15         z = n % 10;
16         printf("%d", z);
17         n=(n - n % 10) / 10;
18         return f(n);
19     }
20 }
21
22 int main() {
23     int n;
24     printf("n = ");
25     scanf("%d", &n);
26     f(n);
27     return 0;
28 }
```

### 3.17.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.15\ reverse\ output.c -o 6.15.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.15.exe
n = 123456789
987654321
Newton@Newton-PC-2 ~/C_Program_Design
$ |

```

### 3.18

写一函数 **digh**( $m, k$ )，它将回送整数  $m$  从右边开始的第  $k$  个数字的值，例如 **digh**(8542, 3) = 5，**digh**(12, 4) = 0。

#### 3.18.1 程序代码

```

1  /*
2  * filename: 6.16 digh.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  int digh(int n, int k) {
9      int y;
10     int t;
11     for (t = 1; t <= k; t++) {
12         y = n % 10;
13         n = (n - y) / 10;
14     }
15     return y;
16 }
17
18 int main() {
19     int n, k;
20     printf("n, k = ");
21     scanf("%d, %d", &n, &k);
22     printf("digh(%d, %d) = %d", n, k, digh(n, k));
23     return 0;
24 }

```

### 3.18.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.16\ digh.c -o 6.16.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.16.exe
n, k = 314159265, 2
digh(314159265, 2) = 6
Newton@Newton-PC-2 ~/C_Program_Design
$ |

```

### 3.19 两数对换

定义一个宏`swap(x, y)`，完成对两个整数 $x, y$ 的交换。

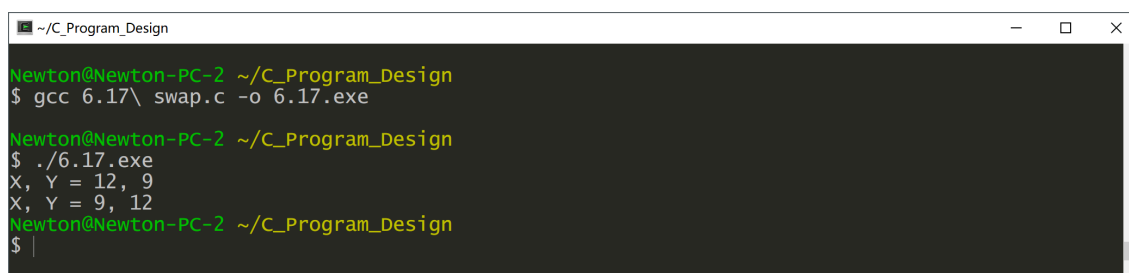
#### 3.19.1 程序代码

```

1  /*
2  * filename: 6.17 swap.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  int X, Y;
9
10 void swap(int x, int y) {
11     Y = x;
12     X = y;
13 }
14
15 int main() {
16     printf("X, Y = ");
17     scanf("%d, %d", &X, &Y);
18     swap(X, Y);
19     printf("X, Y = %d, %d", X, Y);
20     return 0;
21 }

```

#### 3.19.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.17\ swap.c -o 6.17.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.17.exe
X, Y = 12, 9
X, Y = 9, 12
Newton@Newton-PC-2 ~/C_Program_Design
$

```

### 3.20 三数最大值宏

定义一个宏`max(x, y, z)`从三个数 $x, y, z$ 中找出最大数。

#### 3.20.1 程序代码

```

1  /*
2  * filename: 6.18 max.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7

```

```

8  #define MAX(a, b, c)      ((a > b? a : b) > c? (a > b? a : b) : c)
9
10 int main() {
11     int x, y, z;
12     printf("x, y, z = ");
13     scanf("%d, %d, %d", &x, &y, &z);
14     printf("max(%d, %d, %d) = %d", x, y, z, MAX(x, y, z));
15     return 0;
16 }

```

### 3.20.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/_C_Program_Design
$ gcc 6.18\ max.c -o 6.18.exe

Newton@Newton-PC-2 ~/_C_Program_Design
$ ./6.18.exe
x, y, z = 3, 6, 9
max(3, 6, 9) = 9
Newton@Newton-PC-2 ~/_C_Program_Design
$

```

### 3.21 代码分析

先静态阅读以下程序，然后上机运行程序，查看运行结果是否与你阅读的结果一致？不一致的原因何在？

#### 3.21.1 程序代码

```

1  /*
2  * filename: 6.19 analysis.c
3  * property: analysis
4  */
5
6  #include <stdio.h>
7
8  int a = 3, b = 5, c = 2;
9
10 int f(int a, int b) {
11     printf("a = %d, b = %d\n", a, b);
12     a++;
13     b--;
14     printf("a = %d, b = %d\n", a, b);
15     c = a + b;
16     return a + b + c;
17 }
18
19 int main() {
20     int a = 4, b, c, k;
21     printf("a = %d, b = %d, c = %d\n", a, b, c); // added command
22     k = f(a + 2, b + 1);
23     printf("%d, %d, %d, %d", a, b, c, k);
24     return 0;
25 }

```



### 3.21.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 6.19\ analysis.c -o 6.19.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./6.19.exe
a = 4, b = 0, c = 0
a = 6, b = 1
a = 7, b = 0
4, 0, 0, 14
Newton@Newton-PC-2 ~/C_Program_Design
$

```

通过添加几个输出语句，可以大致推断 C 的语言属性：凡是全局变量在某个函数中被重新定义，那就覆盖全局变量。值传递本质上还是一种赋值，被赋值对象也是函数中被定义的变量，如果和全局函数同名必然会将全局函数再度覆盖。如果不覆盖，那就引用全局变量。这个是 C 的 **namespace** 管理机制。

### 3.22

静态分析以下程序的执行结果，然后上机运行程序，将分析结果与运行结果加以对比，从中领会静态局部变量的含义及用法。

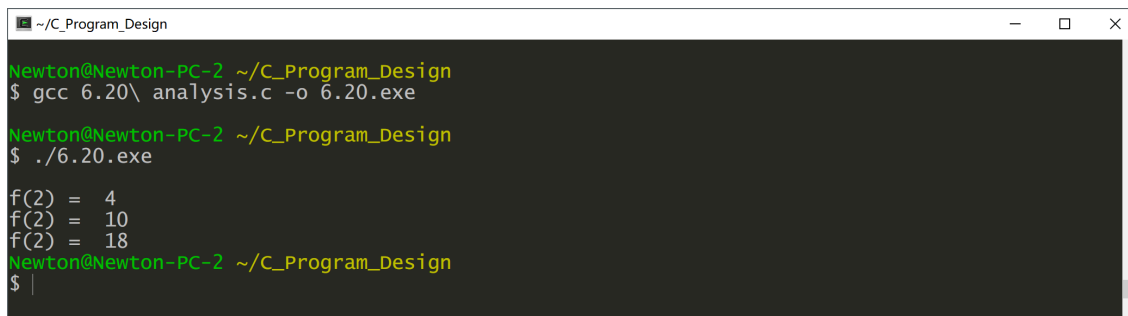
#### 3.22.1 程序代码

```

1  /*
2  * filename: 6.20 analysis.c
3  * property: analysis
4  */
5
6  #include <stdio.h>
7
8  int f(int x) {
9      static int f = 0, y = 0;
10     if (f == 0)
11         y += 2 * x;
12     else
13         if(f == 1)
14             y += 3 * x;
15         else
16             y += 4 * x;
17     f++;
18     return y;
19 }
20
21 int main() {
22     printf("\nf(2) = %d", f(2));
23     printf("\nf(2) = %d", f(2));
24     printf("\nf(2) = %d", f(2));
25
26     return 0;
27 }

```

### 3.22.2 运行结果



```

Newton@Newton-PC-2 ~/_C_Program_Design
$ gcc 6.20\ analysis.c -o 6.20.exe

Newton@Newton-PC-2 ~/_C_Program_Design
$ ./6.20.exe

f(2) = 4
f(2) = 10
f(2) = 18
Newton@Newton-PC-2 ~/_C_Program_Design
$

```

**static** 修饰符可用来修饰函数或者变量。

1. 全局静态变量：在全局变量之前加上关键字 **static**，全局变量就被定义成为一个全局静态变量。
  - (1) 内存中的位置：静态存储区（静态存储区在整个程序运行期间都存在）
  - (2) 初始化：未经初始化的全局静态变量会被程序自动初始化为 0（自动对象的值是任意的，除非他被显示初始化）
  - (3) 作用域：全局静态变量在声明他的文件之外是不可见的。准确地讲从定义之处开始到文件结尾。

定义全局静态变量的好处：

- (1) 不会被其他文件所访问，修改
  - (2) 其他文件中可以使用相同名字的变量，不会发生冲突。
2. 局部静态变量：在局部变量之前加上关键字 **static**，局部变量就被定义成为一个局部静态变量。
  - (1) 内存中的位置：静态存储区
  - (2) 初始化：未经初始化的局部静态变量会被程序自动初始化为 0（自动对象的值是任意的，除非他被显示初始化）
  - (3) 作用域：作用域仍为局部作用域，当定义它的函数或者语句块结束的时候，作用域随之结束。

当 **static** 用来修饰局部变量的时候，它就改变了局部变量的存储位置，从原来的栈中存放改为静态存储区。但是局部静态变量在离开作用域之后，并没有被销毁，而是仍然驻留在内存当中，直到程序结束，只不过我们不能再对他进行访问。当 **static** 用来修饰全局变量的时候，它就改变了全局变量的作用域（在声明他的文件之外是不可见的），但是没有改变它的存放位置，还是在静态存储区中。

## 四、实验总结

当我再次完成这次报告的时候，已经是写过数个体量较大的工程了。不过在做大工程的时候，往往还是把视野局限在完成功能上，对于一些 C 的修饰语句从来都没有使用过，对 C 的内存分配仅是一知半解。经过对本次实验中的几个 `analysis` 属性的代码进行静态分析，我回忆起了往昔一些学过但是不曾用过的东西，这在我做过的很多项目里都是可以起优化作用的。

模块化涉及函数模块化与功能模块化，很多复杂的工程都是把复杂的系统拆分成一个一个的功能模块，源代码放置在不同的文件夹里。这一点在本实验中没有体现出来，我决定放到后期的综合训练中，用 `GNU make` 进行编译管理，具体代码内容的安排，也参见后期的一些工程。

经过对《UNIX 环境高级编程》[1]这本书的学习，还有诸如 *Harley Hahn's Guide to Unix and Linux*[2]这本书的阅读，我觉得基于 Shell 的 UNIX 环境似乎是最适合新手学习的。

本次实验，集中主要精力，在以前版本的基础上，对文档结构进行了重整，看起来自然了很多，目录也规范了很多。有关编程的规范性问题，参考林锐高质量 C/C++编程指南的第一版[3]。

## 五、参考文献

1. Stevens, W.R. and S.A. Rago, *UNIX 环境高级编程*. 2nd ed. 2005, 北京: 人民邮电出版社.
2. Hahn, H., *Harley Hahn's Guide to Unix and Linux*. 2009, New York: McGraw-Hill.
3. 林锐, *高质量 C++/C 编程指南*. 1.0 ed. 2001.

## 六、教师评语