

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：选择结构程序设计		
实验编号：NO.2	实验日期：2018 年 8 月 7 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

1. 进一步掌握运行一个 C 语言程序的方法和步骤。
2. 分清 C 语言的符号、标识符、保留字的区别。
3. 掌握 C 语言的数据类型，会定义整型、实型、字符型变量以及对它们的赋值方法。
4. 学会数据输入方式和数据输出格式及各种格式转意符。
5. 学会使用 C 的运算符以及用这些运算符组成的表达式，特别是自加（++）和自减（--）运算符的使用。

二、实验环境

Windows10 Pro Workstation 17096;
Code::Blocks 16.01 GCC 集成开发环境;
Cygwin GCC 编译器。

三、实验内容

3.1 题 1

输入并运行下面程序，分析其运行结果。这是一段对变量进行不同格式输出打印的程序，在第二次实验里，变量的类型被替换。

3.1.1 程序代码

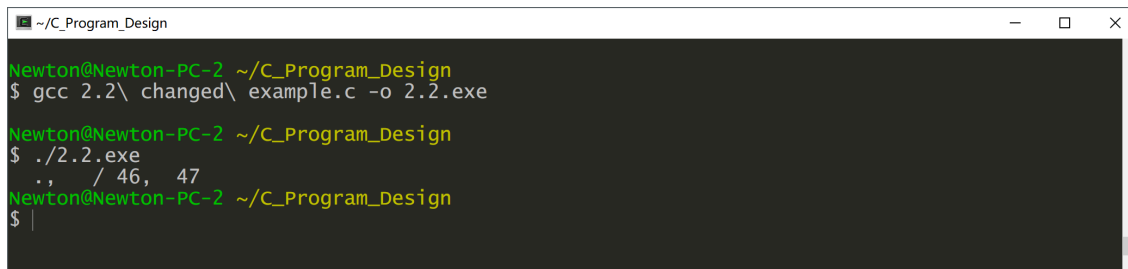
```
1  /*
2  * filename: 2.1 example.c
3  * property: example
4  */
5
6  #include <stdio.h>
7
8  int main()
9  {
10     char c1, c2;
11     c1 = 46;
12     c2 = 47;
13     printf("%3c, %3c", c1, c2);
```

```

14     printf("%3d, %3d", c1, c2);
15     return 0;
16 }

```

3.1.2 运行结果

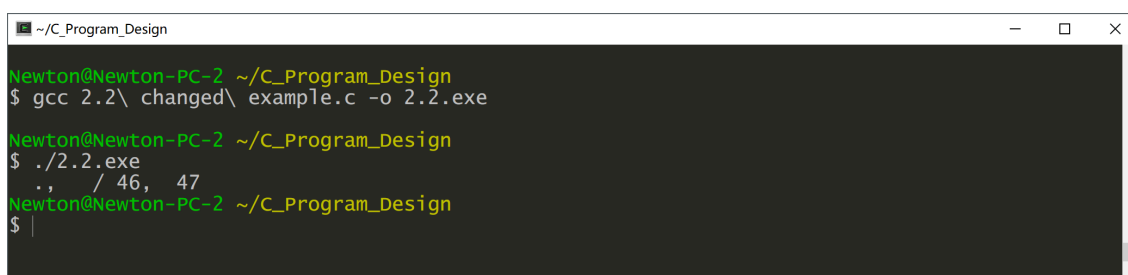


```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.2\ changed\ example.c -o 2.2.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.2.exe
., / 46, 47
Newton@Newton-PC-2 ~/C_Program_Design
$

```

将程序第二行改为: `int c1,c2;`再运行, 分析其结果。



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.2\ changed\ example.c -o 2.2.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.2.exe
., / 46, 47
Newton@Newton-PC-2 ~/C_Program_Design
$

```

注: 实际本例体现出 C 语言的一种特性 (灵活), 整型变量与字符型变量可以相互转换。

3.2 题 2

输入并运行下面程序。运行这个程序, 分析结果, 特别注意输出 `c1`, `c2` 的值是什么? 什么原因? (1) 将输入 `e` 和 `f`, `u` 和 `v` 的语句分别改为:

```

scanf("%d %d",&e, &f);
scanf("%d %d",&u, &v);

```

运行并分析结果。最后, 将程序的第一行加命令:

```
#include <math.h>
```

运行并分析结果。

3.2.1 程序代码

```

1  /*
2  * filename: 2.3 example.c
3  * property: example
4  */
5
6  #include<stdio.h>
7
8  int main() {
9      int a, b;
10     float c, d;
11     long e, f;
12     unsigned int u, v;

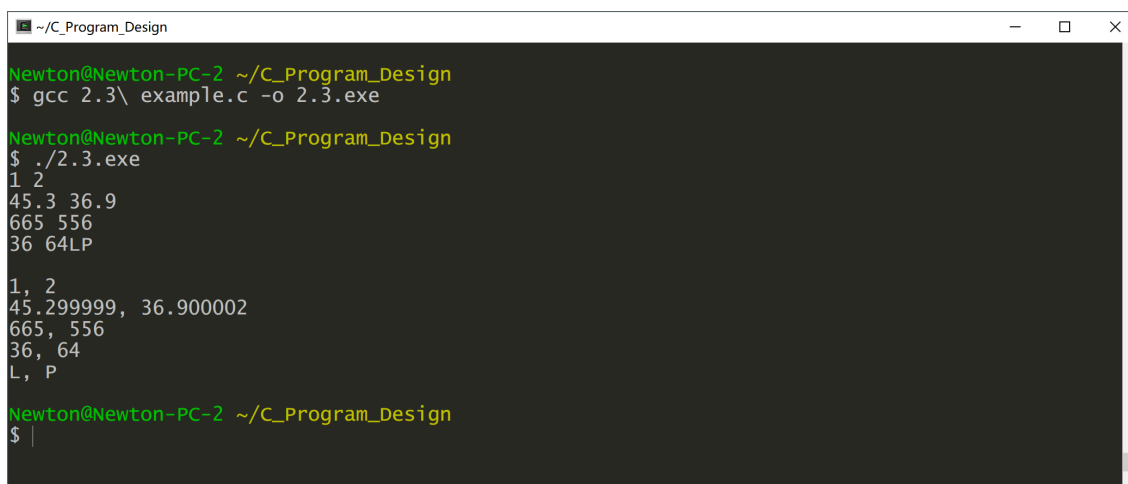
```

```

13     char c1, c2;
14     scanf("%d %d",&a, &b);
15     scanf("%f %f",&c, &d);
16     scanf("%ld %ld",&e, &f);
17     scanf("%o %o",&u, &v);
18     scanf("%c %c",&c1, &c2);
19
20     printf("\n");
21     printf("%d, %d\n",a, b);
22     printf("%f, %f\n",c, d);
23     printf("%ld, %ld\n",e, f);
24     printf("%o, %o\n",u, v);
25     printf("%c, %c\n",c1, c2);
26     return 0;
27 }

```

3.2.2 运行结果




```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.3\ example.c -o 2.3.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.3.exe
1 2
45.3 36.9
665 556
36 64LP
1, 2
45.299999, 36.900002
665, 556
36, 64
L, P
Newton@Newton-PC-2 ~/C_Program_Design
$

```

在输入数字的时候，数字彼此之间可以用空格分割，但是在输入字符的时候，由于空格与回车也算做字符，所以要想输入英文字母之类的，应该不加空格地直接输入。

更改输入方式之后，运行结果如下



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.4\ changed\ example.c -o 2.4.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.4.exe
1 1
1 1
1 1
1 1
1 1LP
1, 1
1.000000, 1.000000
1, 1
1, 1
L, P
Newton@Newton-PC-2 ~/C_Program_Design
$

```

添加 `math` 库之后，运行结果如下：

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.5\ changed\ example.c -o 2.5.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.5.exe
1 2
45.3 36.9
665 556
36 64LP

1, 2
45.299999, 36.900002
665, 556
44, 100
L, P

Newton@Newton-PC-2 ~/C_Program_Design
$

```

可以发现，在这个例子之下，添加数学库之后，运行结果没有发生任何变化。

3.3 题 3

编写一个程序，求表达式 $x - z \% 2 * (x + y) \% 2 / 2$ 的值。设 $x = 8.5$, $y = 2.5$, $z = 4$ 。

3.3.1 程序代码

```

1  /*
2  * filename: 2.6 simple calculation.c
3  * property: homework
4  */
5
6  #include<stdio.h>
7
8  int main() {
9      float x = 8.5;
10     float y = 2.5;
11     float z = 4.0;
12
13     printf("ans = %.4f", x-(int)((int)z % 2 * (double)(int)(x+y)) % 2 / 2);
14     return 0;
15 }

```

3.3.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.6\ simple\ calculation.c -o 2.6.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.6.exe
ans = 8.5000

Newton@Newton-PC-2 ~/C_Program_Design
$

```

3.4 题 4

先分析下面程序的结果，然后再上机运行，看结果上否一致。

3.4.1 程序代码

```

1  /*
2  * filename: 2.7 operator test.c
3  * property: example
4  */
5
6  #include <stdio.h>
7
8  int main() {
9      int x, y, z;
10     x = y = z = 3;
11
12     y = x++ -1; printf("%d,%d\n", x, y);
13     y = ++x -1; printf("%d,%d\n", x, y);
14     y = z-- +1; printf("%d,%d\n", z, y);
15     y = --z +1; printf("%d,%d\n", z, y);
16     return 0;
17 }

```

3.4.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.7\ operator\ test.c -o 2.7.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.7.exe
4,2
5,4
2,4
1,2
Newton@Newton-PC-2 ~/C_Program_Design
$

```

注：本例学生注意，自增自减运算符，先赋值后自增（自减）和先自增（自减）后赋值的问题。

3.5

编写一个程序，将输入的小写字母改写成大写字母并输出。提示：可采用 `getchar()` 函数输入字符，并利用 `for` 循环语句。当然也可用其它方法，只要能够实现其功能即可。下面给出一个语句段，学生补充一个完整的程序后，上机进行调试。

3.5.1 程序代码

```

1  /*
2  * filename: 2.8 lower to capital.c
3  * property: example
4  */
5
6  #include<stdio.h>
7
8  int main() {
9      char c1, c2;

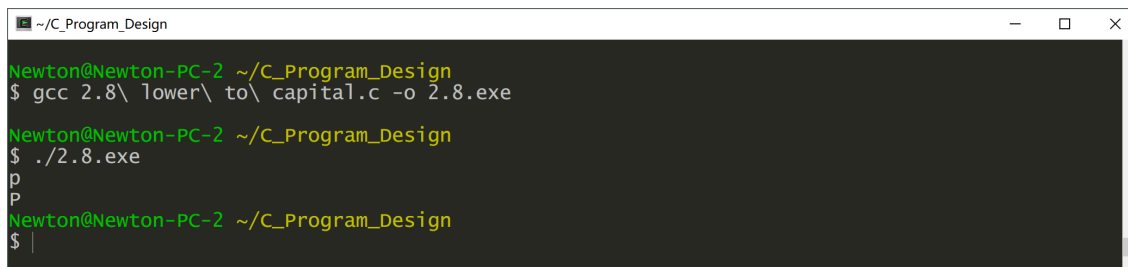
```

```

10    c1 = getchar();
11    c2 = c1 - 32;
12    printf("%c", c2);
13    return 0;
14 }

```

3.5.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/_C_Program_Design
$ gcc 2.8\ lower\ to\ capital.c -o 2.8.exe
Newton@Newton-PC-2 ~/_C_Program_Design
$ ./2.8.exe
p
P
Newton@Newton-PC-2 ~/_C_Program_Design
$

```

3.6 习题[1]

1. (判错) 下述论断哪些是不对的?

- (1) 每个 C 语言程序有且仅有一个主函数 `main()`。
- (2) C 语言程序的每一行都用分号结尾。
- (3) C 程序的执行从第一行开始到最后一行结束。
- (4) C 程序的每一行只能写一条语句。
- (5) C 程序的一条语句可以占多行。
- (6) 一个 C 程序可有一个或多个函数组成。
- (7) 在 C 程序中, 注释说明只能写在一条语句的末尾。
- (8) 在一个 C 程序中, 主函数必须放在程序的首部。
- (9) 在一个 C 程序中, 主函数 `main()` 可以放在程序的任何位置上。
- (10) 在 C 程序中, 注释部分是用花括号括起来的。

3.7 程序查错

3.7.1 (1)

```

1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5      float r, s;
6      s = π * r * r;
7      printf("s = %f\n", s)
8  }

```

答: π 不是 ASCII 字符集里的可用文字。

3.7.2 (2)

运行看看结果如何? 为什么?

3.7.2.1 程序代码

```

1  /*
2  * filename: 2.9 bad code.c
3  * property: example, this code could not run.
4  */
5
6  #include<stdio.h>
7
8  int main() {
9      int i, j, k;
10     float x, y, z;
11     scanf("%d %f %f",&i, &j, &k);
12     scanf("%d %f %f",&x, &y, &z);
13
14     i = i + x;
15     y = y + j;
16     z = i + j;
17     k = x % y;
18
19     printf("%d, %f, %f\n",i, j, k);
20     printf("%f, %d, %d\n",x, y, z);
21     return 0;
22 }
```

3.7.2.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.9\ bad\ code.c -o 2.9.exe
2.9 bad code.c: 在函数 'main' 中:
2.9 bad code.c:16:11: 错误: 双目运算符 % 操作数 ('float'和 'float')无效
    k = x % y;
           ^
Newton@Newton-PC-2 ~/C_Program_Design
$
```

编译器首先会进行报错，第 16 行的求余数运算在浮点型数据上面不能运算。之后有很多警告，不希望对数据进行输入或者输出的时候不按照定义的类型来。因为类型与使用不匹配，所以导致 warning 与 error。

3.7.3 (3)

运行试试看，结果如何？为什么？

3.7.3.1 程序代码

```

1  /*
2  * filename: 2.10 type test.c
3  * property: example, this code could not run.
4  */
5
6  #include<stdio.h>
7
```

```

8  int main() {
9      int i = 100;
10     float x = 200;
11     long y = 300;
12     printf("i = %d, x = %d, y = %d\n", i, x, y);
13     printf("i = %f, x = %f, y = %f\n", i, x, y);
14     printf("i = %ld, x = %ld, y = %ld\n", i, x, y);
15     return 0;
16 }

```

3.7.3.2 运行结果

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.10\ type\ test.c -o 2.10.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.10.exe
i = 100, x = 0, y = 300
i = 0.000000, x = 200.000000, y = 0.000000
i = 100, x = 4641240890982006784, y = 300

Newton@Newton-PC-2 ~/C_Program_Design
$

```

浮点型数据与整型数据的存储方式不一样，强行进行打印，就会出现内存读取与设想的不一样。打印和运算不一样，运算会进行一些类型转换，但是打印则不同。

3.8 题 5

参考下面程序，如何改写输入函数，并配合正确的键盘输入方法才能使 `x`、`y` 和 `ch` 分别获得值 10，100 和 'A'？若将输入语句改为 `scanf("%d,%d,%c",&x,&y,&ch);` 或者 `scanf("%d,%c,%d",&x,&ch,&y);` 或者 `scanf("%c,%d,%d",&ch,&x,&y);` 结果将会如何？

```

1  /*
2   * filename: 2.11 input test.c
3   * property: example
4   */
5
6  #include<stdio.h>
7
8  int main() {
9      int x, y;
10     char ch;
11     scanf("x = %d, y = %d, ch = %c", &x, &y, &ch);
12     printf("x = %d, y = %d, ch = %c\n", x, y, ch);
13     return 0;
14 }

```

答：键盘输入的方法是在对应位置上输入同样的字符，如果不改变 `scanf` 函数引号里的内容，就要输入 `x=3, y=2, ch=1`（其他的数字也好，按自己的需要来。）

3.9

下面程序对输入有何要求？利用它可以作什么？

3.9.1 程序代码

```

1  /*
2  * filename: 2.12 print test.c
3  * property: example
4  */
5
6  #include<stdio.h>
7
8  int main() {
9      int x, y, z;
10     long m;
11
12     scanf("%d %o %x", &x, &y, &z);
13     scanf("%ld", &m);
14
15     printf("x=%d, %o, %x\n", x, x, x);
16     printf("y=%d, %o, %x\n", y, y, y);
17     printf("z=%d, %o, %x\n", z, z, z);
18     printf("m=%ld, %lo, %lx\n", m, m, m);
19     return 0;
20 }

```

```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.12\ print\ test.c -o 2.12.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.12.exe
1000 100000 11222 565
x=1000, 1750, 3e8
y=32768, 100000, 8000
z=70178, 211042, 11222
m=565, 1065, 235
Newton@Newton-PC-2 ~/C_Program_Design
$

```

可以通过这种方式进行进位制转化。

3.10 补码

你记得补码¹是何意义吗？下面的程序说明了什么？如果将 x , y 各赋值为 -1 。或者各赋值为 -32768 和 -2147483648 ，输出结果如何？为什么？

```

1  /*
2  * filename: 2.13 memory test.c
3  * property: example
4  */
5
6  #include<stdio.h>
7
8  int main() {
9      int x = -500;

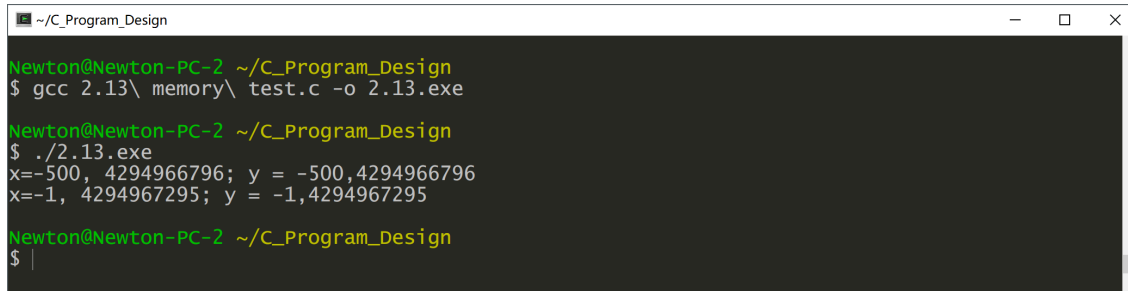
```

¹ 补码，整数 X 补码指的是：对于正数与原码相同；对于负数，数符位为1，数值位为 X 的绝对值取反后加1，即为反码加1。

```

10     long y = -500;
11     printf("x=%d, %u; y = %ld,%u\n", x, x, y, y);
12     x = -1;
13     y = -1;
14     printf("x=%d, %u; y = %ld,%u\n", x, x, y, y);
15
16     return 0;
17 }

```



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.13\ memory\ test.c -o 2.13.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.13.exe
x=-500, 4294966796; y = -500,4294966796
x=-1, 4294967295; y = -1,4294967295

Newton@Newton-PC-2 ~/C_Program_Design
$

```

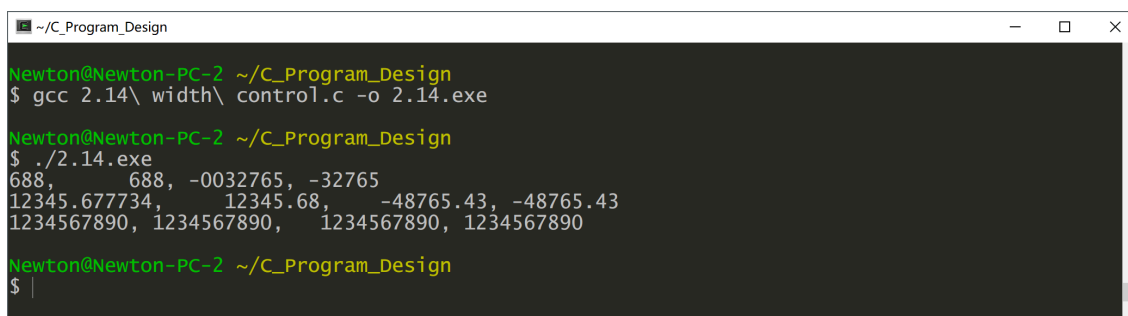
这个涉及数据的存储形式，如果都改成 -1 ，那么根据无符号类型的定义， -1 的 32 位补码是 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111，输出无符号，那么认定之为正数，转换成十进制就是 4294967295，即 $2^{32} - 1$ 。

3.11 输出宽度及控制

```

1  /*
2  * filename: 2.14 width control.c
3  * property: example
4  */
5
6  #include<stdio.h>
7
8  int main() {
9      int i, j;
10     float x, y;
11     long int m;
12     i = 688;
13     j = -32765;
14     x = 12345.678;
15     y = -48765.432;
16     m = 1234567890;
17
18     printf("%d, %8d, %08d, %-8d\n", i, i, j, j);
19     printf("%f, %12.2f, %12.2f, %-12.2f\n", x, x, y, y);
20     printf("%ld, %lu, %12ld, %-12ld\n", m, m, m, m);
21     return 0;
22 }

```



```
~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 2.14\ width\ control.c -o 2.14.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./2.14.exe
688,          688, -0032765, -32765
12345.677734,    12345.68,    -48765.43, -48765.43
1234567890, 1234567890,    1234567890, 1234567890

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

可以发现，`%08d` 中，`0` 的作用就是将不足八位的地方，在高位补上 `0`；而 `%-8d` 中负号的作用是改变数据输出的偏向规则。正常是居右显示，加了负号就是往左靠，不足的地方补空格。

四、 实验总结

通过调试例题的程序，思考所提出的问题，收获了关于数据的存储、输入以及输出的一些问题的解决方案。

五、 参考文献

1. Prata, S., *C Primer Plus* 中文版. 6th ed. C 和 C++ 实务精选. 2016, 北京: 人民邮电出版社.

六、 教师评语