

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：指针程序设计(1)		
实验编号：No.07	实验日期：2017 年 8 月 20 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. 了掌握变量的指针及其基本用法。
- 2. 掌握一维数组的指针及其基本用法。
- 3. 掌握指针变量作为函数的参数时，参数的传递过程及其用法。

二、实验环境

Windows10 Pro Workstation 17134.228;  
Cygwin GCC 编译器。

三、实验内容

3.1 指针值观测

对以下程序进行单步运行,并从中了解变量的指针和指针变量的概念。

3.1.1 程序代码

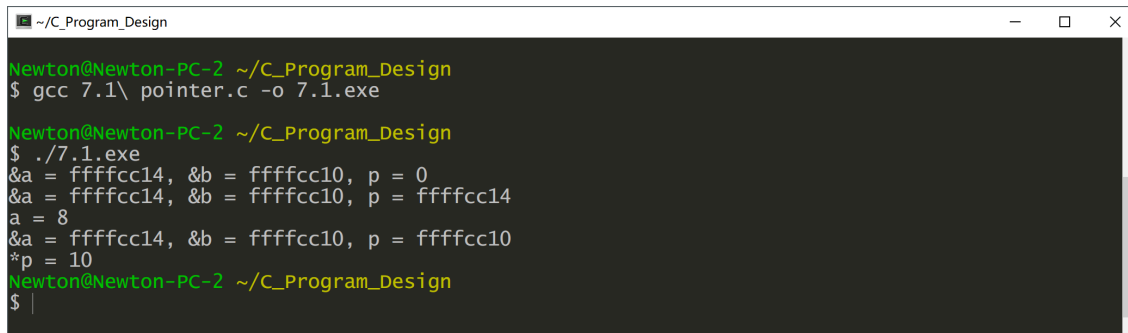
```
1  /*
2   * filename: 7.1 pointer.c
3   * property: test
4   */
5
6  #include <stdio.h>
7
8  int main() {
9      int a = 5, b = 5, *p;
10     printf("&a = %x, &b = %x, p = %x\n", &a, &b, p);
11     p = &a;
12     *p = 8;
13     printf("&a = %x, &b = %x, p = %x\n", &a, &b, p);
14     printf("a = %d\n", a);
15     p = &b;
16     b = 10;
17     printf("&a = %x, &b = %x, p = %x\n", &a, &b, p);
```

```

18     printf("*p = %d", *p);
19
20     return 0;
21 }

```

### 3.1.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 7.1\ pointer.c -o 7.1.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./7.1.exe
&a = fffffcc14, &b = fffffcc10, p = 0
&a = fffffcc14, &b = fffffcc10, p = fffffcc14
a = 8
&a = fffffcc14, &b = fffffcc10, p = fffffcc10
*p = 10
Newton@Newton-PC-2 ~/C_Program_Design
$

```

按以下步骤操作：

- ② 输入程序后，连续按三次 F8，使绿条停留在 `P=&a` 语句行上。
- ② 用<Ctrl F7>操作分别将 `a`，`b`，`&a`，`&b`，`p` 及 `*p` 显示出来。
- ③ 查看观察窗口中的内容，可发现此时 `a`、`b` 已有确定的地址(`&a` 和 `&b`)和确定的值，而 `p` 还没有确定的值(此时语句 `p=&a` 还未执行)，即 `p` 还没有明确的指向，因而它所指向的内存单元(`*p`)中的内容也是不确定的。
- ④ 按 F8 往下执行一步后再查看观察窗中的内容，可发现 `p` 已有确定的值，它与 `&a` 的值一致，说明 `p` 中存放了变量 `a` 的地址，也就是说 `p` 是指向变量 `a` 的指针变量。同时可发现，`*p` 的内容与 `a` 的内容一致，即 `p` 所指向的内存单元中的内容就是 `a` 的内容；从而可以理解 `*p` 等效于 `a`，表示同一内存单元。
- ⑤ 按 F8 往下执行一步后再查看观察窗中的内容，可发现 `*p` 和 `a` 的内容都已发生变化，从而可理解通过改变指针变量 `p` 所指向的内存单元中的内容可以间接地改变 `a` 中的内容。
- ⑥ 再按 F8 往下执行一步，可发现 `p` 的值已发生变化，它与 `&b` 的值一致，说明 `p` 已经是指向变量 `b` 的指针变量，它不再是指向 `a`，`*p` 的内容也已变为 `b` 的内容，从而可理解指针变量的指向是随时可以改变的。
- ⑦ 再按 F8 往下执行一步，可发现，`b` 的值和 `*p` 的值都已发生变化，即改变 `b` 的内容就等于改变指针变量 `p` 所指向的内存单元中的内容。

### 3.2 指针值与指向值观测

单步运行以下程序，观察 `&a[0]`，`&a[i]` 和 `P` 的变化，然后回答以下问题：

- ① 程序的功能是什么？  
答：求数组元素的和。
- ② 在开始进入循环体之前，`p` 指向谁？  
答：指向数组的首地址。
- ③ 循环每增加一次，`p` 的值(地址)增加多少？它指向谁？  
答：增加 `sizeof(int)`；指向数组的下一个元素。
- ④ 退出循环后，`p` 指向谁？  
答：指向内存中，数组变量最后一个元素的后一个变量。
- ⑤ 你是否初步掌握了通过指针引用数组元素的方法？

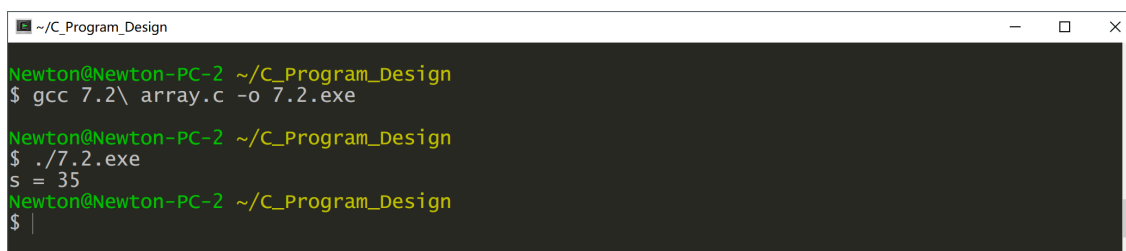
```
1  /*
```

```

2  * filename: 7.2 array.c
3  * property: test
4  */
5
6  #include <stdio.h>
7
8  int main() {
9      int i, *p, s = 0;
10     int a[5] = {5, 6, 7, 8, 9};
11
12     p = a;
13
14     for(i = 0; i < 5; i++, p++) {
15         s += *p;
16     }
17     printf("s = %d", s);
18     return 0;
19 }

```

### 3.2.1 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 7.2\ array.c -o 7.2.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./7.2.exe
s = 35
Newton@Newton-PC-2 ~/C_Program_Design
$

```

## 3.3 指针引用练习

先分析以下程序的运行结果，然后上机验证，并通过此例掌握通过指针变量引用数组元素的各种方法。

### 3.3.1 程序代码

```

1  /*
2  * filename: 7.3 analysis.c
3  * property: analysis
4  */
5
6  #include <stdio.h>
7
8  int main() {
9      int i, s1 = 0, s2 = 0, s3 = 0, s4 = 0, *p;
10     int a[5] = {1, 2, 3, 4, 5};
11     p = a;
12     for (i = 0; i < 5; i++) {
13         s1 += p[i];
14     }
15     for (i = 0; i < 5; i++) {
16         s2 += *(p + i);

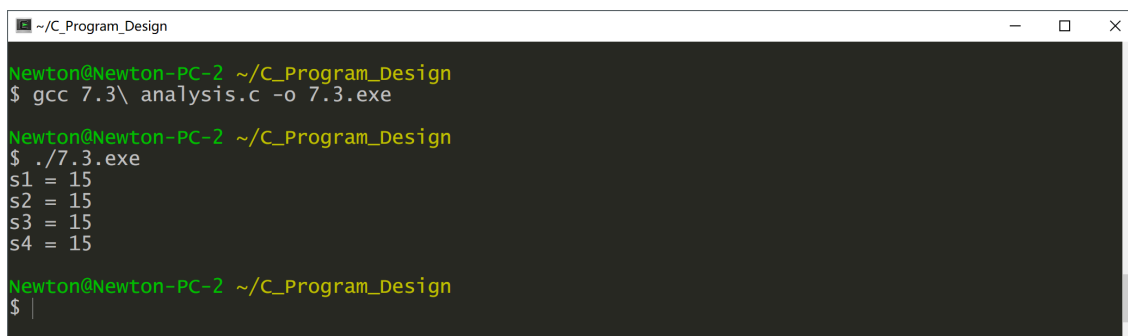
```

```

17     }
18     for (p = a; p < a + 5; p++) {
19         s3 += *p;
20     }
21     p = a;
22     for (i = 0; i < 5; i++) {
23         s4 += *p++;
24     }
25     printf("s1 = %d\ns2 = %d\ns3 = %d\ns4 = %d\n", s1, s2, s3, s4);
26     return 0;
27 }

```

### 3.3.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 7.3\ analysis.c -o 7.3.exe
Newton@Newton-PC-2 ~/C_Program_Design
$ ./7.3.exe
s1 = 15
s2 = 15
s3 = 15
s4 = 15
Newton@Newton-PC-2 ~/C_Program_Design
$

```

## 3.4 一元二次方程求根

编写函数实现计算一元二次方程的两个实根，然后编写主函数调用此函数。

要求：在同一个函数内求出方程的两个实根，此函数不准使用全局变量进行数据传递，也不能使用 `return` 语句，只能通过指针进行数据传递。

### 3.4.1 程序代码

```

1  /*
2  * filename: 7.4 root.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7  #include <math.h>
8
9  void root(float *a, float *b, float *c, float *x1, float *x2) {
10     float d = (*b) * (*b) - 4 * (*a) * (*c);
11     if (d < 0) {
12         printf("no real root;");
13     }
14     else {
15         *x1 = (-(*b) + sqrt(d)) / (2 * (*a));
16         *x2 = (-(*b) - sqrt(d)) / (2 * (*a));
17     }
18 }

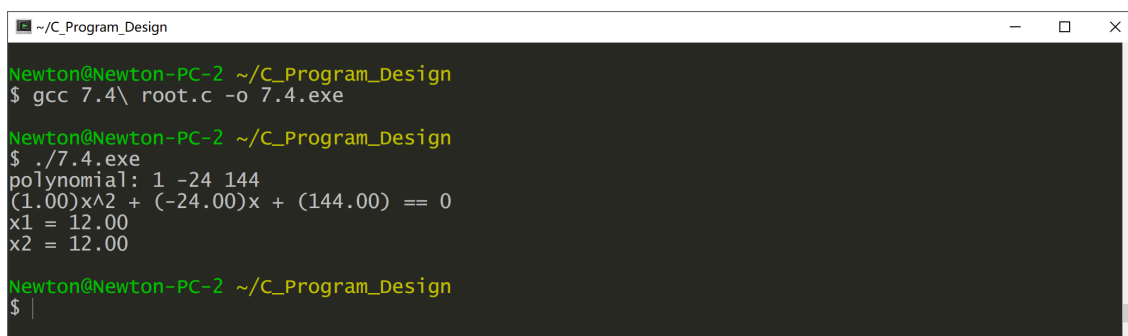
```

```

19
20 int main() {
21     printf("polynomial: ");
22     float a, b, c;
23     float x1, x2;
24     scanf("%f %f %f", &a, &b, &c);
25
26     printf("(%.2f)x^2 + (%.2f)x + (%.2f) == 0\n", a, b, c);
27
28     root(&a, &b, &c, &x1, &x2);
29
30     printf("x1 = %.2f\nx2 = %.2f\n", x1, x2);
31
32     return 0;
33 }

```

### 3.4.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 7.4\ root.c -o 7.4.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./7.4.exe
polynomial: 1 -24 144
(1.00)x^2 + (-24.00)x + (144.00) == 0
x1 = 12.00
x2 = 12.00

Newton@Newton-PC-2 ~/C_Program_Design
$

```

## 3.5 逆序输出序列

编写函数，将 $n$ 个数按原来的顺序的逆序排列（要求用指针实现），然后编写主函数完成：

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

### 3.5.1 程序代码

```

1  /*
2  * filename: 7.5 reverse output.c
3  * property: exercise
4  */
5
6  #include <stdio.h>
7
8  #define NUM      256
9
10 void reverse(int *a, int count) {
11     int i;
12     int tmp;
13     for (i = 0; i <= count / 2; i++) {

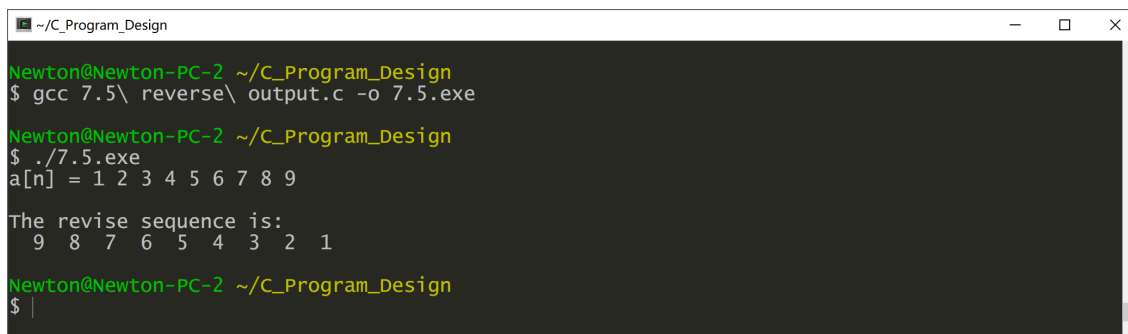
```

```

14     tmp = a[i];
15     a[i] = a[count - i];
16     a[count - i] = tmp;
17 }
18 }
19
20 int main() {
21     int i, a[NUM];
22     int *p, count = 0;
23     p = a;
24     char c;
25     printf("a[n] = ");
26     for(i = 0; i < NUM; i++) {
27         scanf("%d%c", &a[i], &c);
28         if (c != ' ') {
29             break;
30         }
31         else {
32             count += 1;
33         }
34     }
35
36     reverse(a, count);
37
38     printf("The revise sequence is:\n");
39     for(i = 0; i <= count; i++) {
40         printf("%3d", *(p + i));
41     }
42     printf("\n");
43     return 0;
44 }

```

### 3.5.2 运行结果



```

~/C_Program_Design
Newton@Newton-PC-2 ~/C_Program_Design
$ gcc 7.5\ reverse\ output.c -o 7.5.exe

Newton@Newton-PC-2 ~/C_Program_Design
$ ./7.5.exe
a[n] = 1 2 3 4 5 6 7 8 9

The revise sequence is:
  9  8  7  6  5  4  3  2  1

Newton@Newton-PC-2 ~/C_Program_Design
$

```

## 四、实验总结

指针是高级数据结构的基石。通过指针，可以更加方便地对内存中的数据进行操作，写出比较好的分离式模块。通过迭代器模式，可以将算法与数据结构分开，而这也是通过指针来实现的。

随着时间的推移，我的编译器选择历经轮转，从最初的古老的 TC2.0，到 Code::Blocks 集成开发环境，

再到 Visual Studio 2017，最终还是回到了 GNU 平台上来，使用开源的一套库进行实验。在此期间，云南大学也从一个普通的 211 大学跻身双一流大学行列，高级语言程序设计这门课程是否也该升级一下？TC2.0 这个编译器，界面十分古朴，在几十年前绝对算是一流的软件，但是现在，确实落后了，这主要是因为它无法编译在 64 位系统下运行的程序。但是 TC2.0 有很多的优势，比如完全可视化的编译过程，不会生成很多附加文件，这一点 Visual Studio 就太过专业化。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