

華中科技大學

课程实验报告

课程名称: C 语言程序设计实验

专业班级: _____

学 号: _____

姓 名: _____

指导教师: _____

报告日期: _____

软件学院

目 录

1	表达式和标准输入与输出实验.....	1
1.1	实验目的.....	1
1.2	实验内容.....	1
1.3	实验小结.....	9
2	流程控制实验.....	10
2.1	实验目的.....	10
2.2	实验内容及要求.....	10
2.3	实验小结.....	23
3	函数与程序结构实验.....	24
3.1	实验目的.....	24
3.2	实验内容.....	24
3.3	实验小结.....	40
4	编译预处理实验.....	41
4.1	实验目的.....	41
4.2	实验内容.....	41
4.3	实验小结.....	51
5	编译预处理实验.....	53
5.1	实验目的.....	53
5.2	实验内容及要求.....	53
5.3	实验小结.....	68
6	指针实验.....	69
6.1	实验目的.....	69
6.2	实验内容及要求.....	69
6.3	实验小结.....	88
7	结构与联合实验.....	90
7.1	实验目的.....	90
7.2	实验题目及要求.....	90
7.3	实验小结.....	106
8	文件操作实验.....	107
8.1	实验目的.....	107
8.2	实验题目及要求.....	107
8.3	实验小结.....	114

实验 1 表达式和标准输入与输出实验

1.1 实验目的

(1) 熟练掌握各种运算符的运算功能，操作数的类型，运算结果的类型及运算过程中的类型转换，重点是 C 语言特有的运算符，例如位运算符，问号运算符，逗号运算符等；熟记运算符的优先级和结合性。

(2) 掌握 `getchar`, `putchar`, `scanf` 和 `printf` 函数的用法。

(3) 掌握简单 C 程序（顺序结构程序）的编写方法。

1.2 实验内容

1 源程序改错

下面给出了一个简单 C 语言程序例程，用来完成以下工作：

(1) 输入华氏温度 `f`，将它转换成摄氏温度 `c` 后输出；

(2) 输入圆的半径值 `r`，计算并输出圆的面积 `s`；

(3) 输入短整数 `k`、`p`，将 `k` 的高字节作为结果的低字节，`p` 的高字节作为结果的高字节，拼成一个新的整数，然后输出；

在这个例子程序中存在若干语法和逻辑错误。要求参照 1.3 和 1.4 的步骤对下面程序进行调试修改，使之能够正确完成指定任务。

```
1.  #include<stdio.h>
2.  #define PI 3.14159;
3.  void main( void )
4.  {
5.      int f ;
6.      short p, k ;
7.      double c , r , s ;
8.      /* for task 1 */
9.      printf("Input  Fahrenheit:");
10.     scanf("%d", f ) ;
11.     c = 5/9*(f-32) ;
12.     printf( "\n %d (F) = %.2f (C)\n\n", f, c ) ;
13.     /* for task 2 */
14.     printf("input the radius r:");
```

```

15.     scanf("%f", &r);
16.     s = PI * r * r;
17.     printf("\nThe acreage is %.2f\n\n",&s);
18.     /* for task 3 */
19.     printf("input hex int k, p :");
20.     scanf("%x %x", &k, &p );
21.     newint = (p&0xff00)|(k&0xff00)<<8;
22.     printf("new int = %x\n\n",newint);
23. }

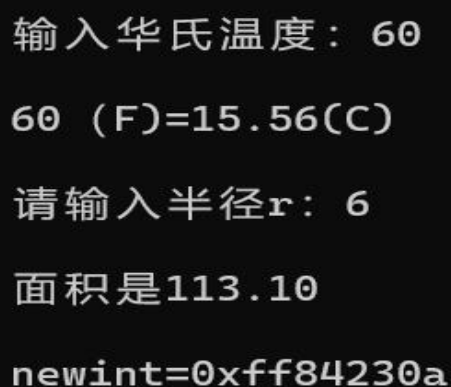
```

解答：

(1) 错误修改：

- 1) 第 2 行的符号常量定义后不能有分号，正确形式为：`#define PI 3.14159`
- 2) 第 5 行的 f 类型错误，正确形式为：`double f ;`
- 3)第 10 行的格式符错误，正确形式为：`scanf("%lf",&f);`
- 4)第 11 行 c 的类型为 double，正确形式为：`c = 5.0/9 * (f-32) ;`
- 5)第 12 行格式符错误，正确形式为：`printf("\n %.2lf (F) = %.2lf (C)\n\n", f, c) ;`
- 6)第 15 行格式符错误，正确形式为：`scanf("%lf",&r);`
- 7)第 17 行 printf 用法错误，输出不应是地址，正确形式为：`printf("\nThe acreage is %.2lf\n\n",s)`
- 8)第 21 行使用位声明变量，且算法错误，正确形式为：`int newint = (p&0xff00) | ((k&0xff00)>>8) ;`

(2) 错误修改后运行结果：



```

输入华氏温度： 60
60 (F)=15.56(C)
请输入半径r： 6
面积是113.10
newint=0xff84230a

```

图 1-1 错误修改后的程序运行结果示意图

2 程序设计

(1) 输入字符 c ，如果 c 是大写字母，则将 c 转换成对应的小写，否则 c 的值不变，输入 Ctrl+Z 程序结束。要求①用条件表达式；②字符的输入输出用 `getchar` 和 `putchar` 函数。程序应能循环接受用户的输入，直至输入 Ctrl+Z 程序结束。

解答：

1) 算法流程如图 1.2 所示

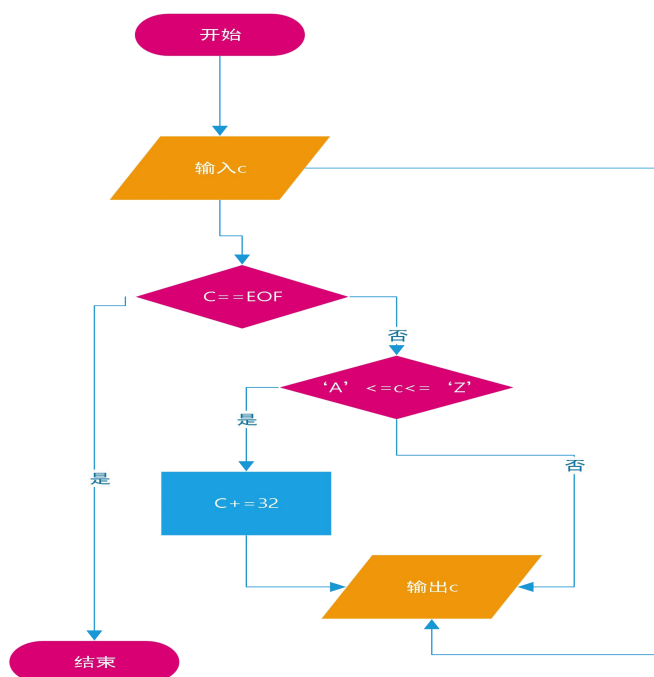


图 1-2 编程题 1 的程序流程图

2) 程序清单

```

1.  #include<stdio.h>
2.  int main()
3.  {
4.      char c;
5.      while((c=getchar())!=EOF)
6.      {
7.          if(c>='A'&&c<='Z')//检验范围
8.              putchar(c+32);//根据ASCII表，大小写字母差值为32
9.          else
10.             putchar(c);
11.     }
12.     return 0;
13. }
  
```

3) 测试

(a) 测试数据:

表 1-1 编程题 1 的测试数据

测试用例	程序输入	理论结果
用例 1	A	a
用例 2	b	b
用例 3	C	c
用例 4	Ctrl+Z	程序退出

(b) 对应测试数据的运行结果截图

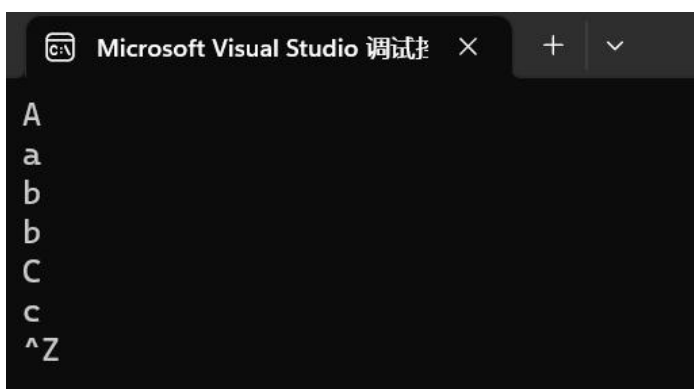


图 1-3 编程题 1 的测试用例的运行结果

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) 输入无符号短整数 x , m , n ($0 \leq m \leq 15$, $1 \leq n \leq 16-m$), 取出 x 从第 m 位开始向左的 n 位 (m 从右至左编号为 $0 \sim 15$), 并使其向左端 (第 15 位) 靠齐。要求: ①检查 m 和 n 的范围; ② x 的值以十六进制输入, m 和 n 以十进制输入; ③结果以十六进制输出。

解答:

1) 算法流程如图 1.4 所示

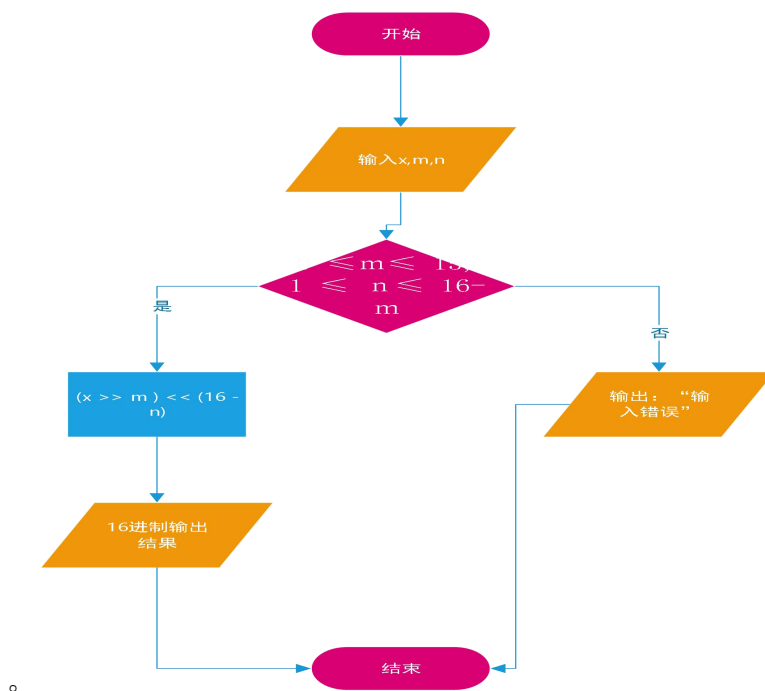


图 1-4 编程题 2 的程序流程图

2) 程序清单

```

1.  #include <stdio.h>
2.  int main() {
3.      unsigned short x;
4.      unsigned short m, n;
5.      scanf("%hx%hd%hd", &x,&m,&n);
6.      if(m>15 || m<0 || n<1 || n>16-m)
7.      {
8.          printf("输入错误! ");
9.      }
10.     else{
11.
12.         unsigned short a = (x>>(m-1)); // 顶到最右边
13.         unsigned short result = (a<<(15-n)); // 向左靠齐
14.
15.         printf("%hx", result);}
16.
17.     return 0;
18. }
  
```

3) 测试

(a) 测试数据:

表 1-2 编程题 2 的测试数据

测试用例	程序输入			理论结果	运行结果
	X	m	N		
用例 1	0100 0110 1000 0000 (4680)	7	4	计算结果 1101 0000 0000 0000 即 D000	D000
用例 2	1101 0101 1000 0011 (D583)	16	1	输入错误 (m 值超范围)	输入错误 (m 值超范围)
用例 3	1101 0101 1000 0011 (D583)	13	5	输入错误 (n 值超范围)	输入错误 (m 值超范围)

(b) 对应测试用例 1 的运行结果如图 1-5 所示。

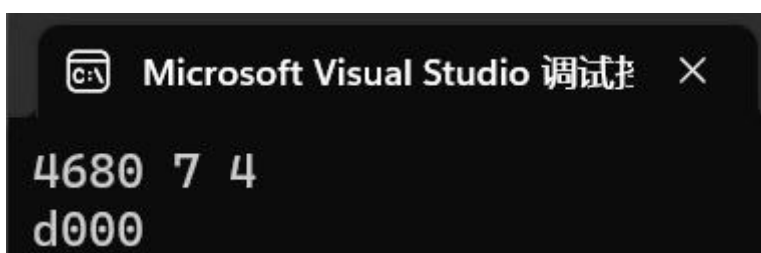


图 1-5 编程题 2 的测试用例一的运行结果

对应测试用例 2 的运行结果如图 1-6 所示。

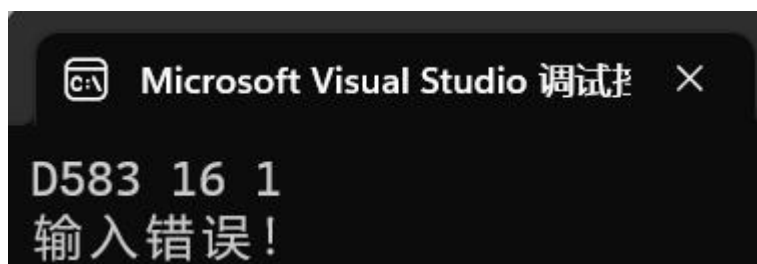


图 1-6 编程题 2 的测试用例二的运行结果

对应测试用例 3 的运行结果如图 1-7 所示

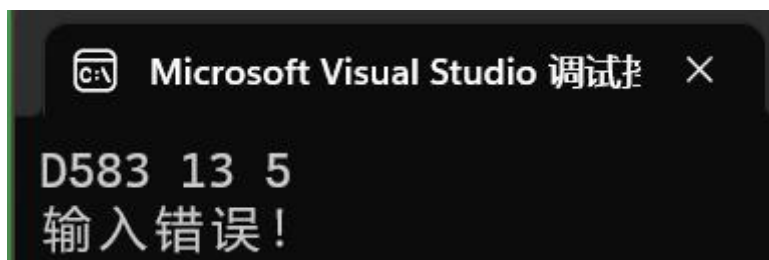


图 1-7 编程题 2 的测试用例三的运行结果

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) IP 地址通常是 4 个用句点分隔的小整数（即点分十进制），但这些地址在机器中是用一个无符号长整型数表示的。例如 3232235876，其机内二进制表示就是 11000000 10101000 00000001 01100100，按照 8 位一组用点分开，该 IP 地址就写成 192.168.1.100。

读入无符号长整型数表示的互联网 IP 地址，对其译码，以常见的点分十进制形式输出。要求循环输入和输出，直至输入 Ctrl+Z 结束。

解答：

1) 算法流程如图 1.8 所示。

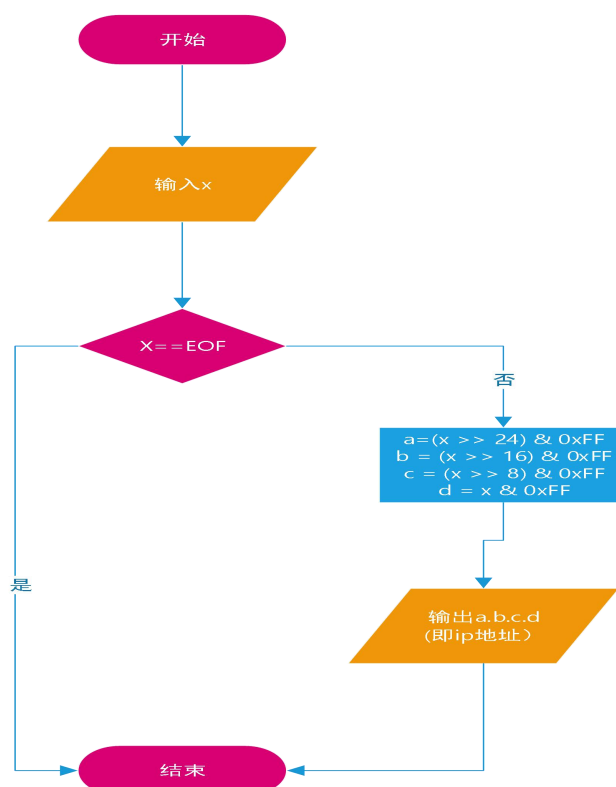


图 1-8 编程题 3 的程序流程图

2) 源程序清单

```

1.     #include <stdio.h>
2.
3.     int main()
4.     {
5.         int x;
6.         while (scanf("%d", &x) != EOF) {

```

```
7.  
8.         int a = (x >> 24) & 0xFF; // 第一个八位  
9.         int b = (x >> 16) & 0xFF; // 第二个八位  
10.        int c = (x >> 8) & 0xFF;  // 第三个八位  
11.        int d = x & 0xFF;         // 第四个八位  
12.        printf("%d.%d.%d.%d\n", a, b, c, d);  
13.    }  
14.    return 0;  
15. }
```

3) 测试

(a) 测试数据:

表 1-3 编程题 3 的测试数据

测试用例	程序输入	理论结果
用例 1	3232235876	192.168.1.100
用例 2	66322345376	113.30.173.160
用例 3	187232376	11.40.240.120

b) 对应测试数据的运行结果截图

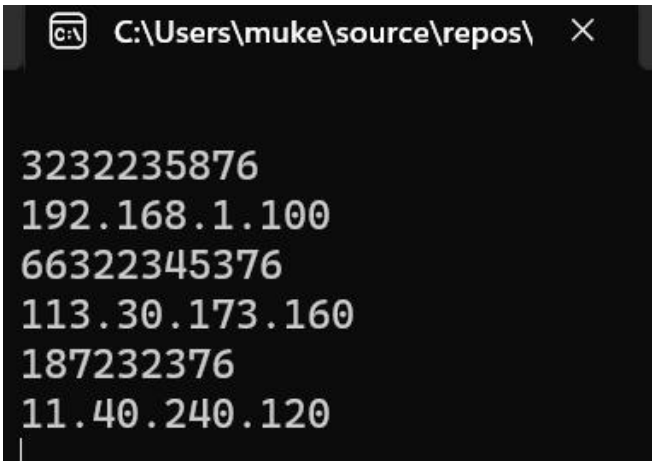


图 1-9 编程题 3 的测试用例的运行结果

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。

1.3 实验小结

在完成这次实验后，我获得了以下几点体会和经验：

1. **重视细节，善于纠错：** 在 C 语言中，细节非常重要。一个小小的语法错误或逻辑错误都可能导致程序无法正常工作。通过这次实验，我了解了各种常见错误的原因和报错显示，例如全半角逗号分号会报错未识别字符，scanf 后面的变量忘记加&会报错 C4477, #define 后加分号可能导致后面变量被识别为指针，VS2022 需要使用 scanf_s 以保证安全性等等。
2. **输入输出函数的熟练使用：** scanf 和 printf 是 C 语言中常用的输入输出函数。通过实验，我熟悉了几种常见格式字符的用法，和两个函数的使用方法
3. **程序结构：** 这个实验强调了顺序结构程序的编写。在实际编程中，理解程序的结构和流程对于正确完成任务非常重要。同时，实验还涉及了循环输入，通过实验，我了解了 while 的用法和 EOF 的概念，学会如何循环输入和结束输入，对程序的输入输出有了更深的了解。
4. **流程图的画法：** 通过实验，我了解 visio 软件的基本使用，熟悉了用流程图表示程序结构，整理思路。

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 实验内容及要求

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*/

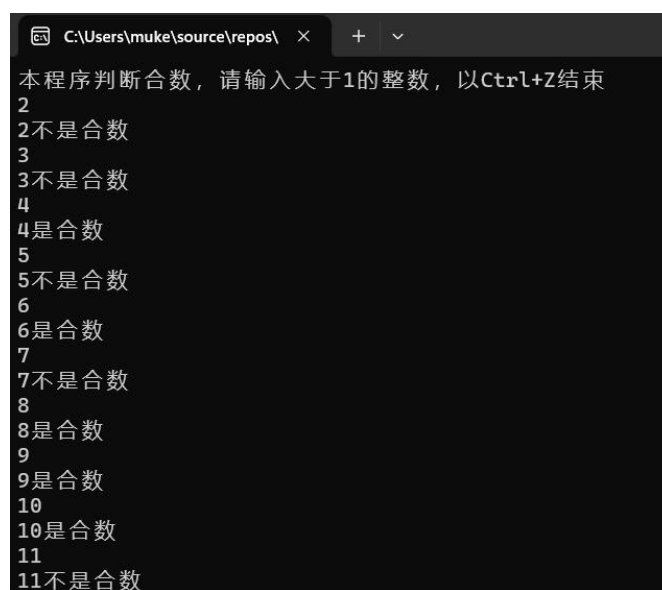
```
1.  #include <stdio.h>
2.  int main( )
3.  {
4.  int i, x, k, flag = 0;
5.  printf("本程序判断合数, 请输入大于 1 的整数, 以 Ctrl+Z 结束\n");
6.  while (scanf("%d", &x) != EOF) {
7.  for(i=2, k=x>>1; i<=k; i++)
8.  if (!x%i) {
9.  flag = 1;
10. break;
11. }
12. if(flag=1) printf("%d 是合数", x);
13. else printf("%d 不是合数", x);
14. }
15. return 0;
16. }
```

解答：

(1) 错误修改:

- 1) 第 4 行 flag 定义为 0 后在循环开始时没有归 0, 正确形式为: 在第六行后加语句 `flag = 0;`
- 2) 第 8 行运算优先级错误, 正确形式为: `if (!(x % i))`
- 3) 第 12 行运算符错误, 不应为赋值符, 正确形式为: `if (flag == 1)`

(2) 错误修改后运行结果:



```

C:\Users\muke\source\repos\
本程序判断合数, 请输入大于1的整数, 以Ctrl+Z结束
2
2不是合数
3
3不是合数
4
4是合数
5
5不是合数
6
6是合数
7
7不是合数
8
8是合数
9
9是合数
10
10是合数
11
11不是合数
    
```

图 2-1 错误修改后的程序运行结果示意图

2. 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:

程序等价于当 `flag=1` 后退出循环, 可以在 `for` 循环中添加一个判断条件, 因此, 题目 (1) 替换后的程序如下所示:

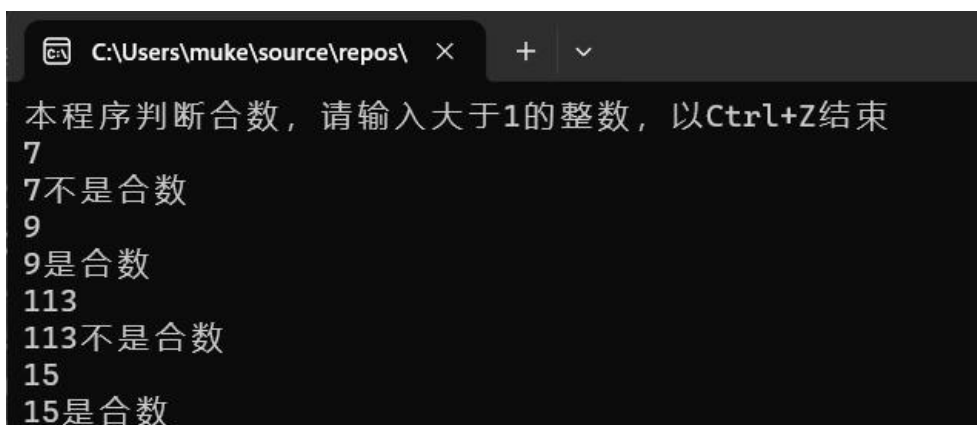
```

1.  #include <stdio.h>
2.  int main()
3.  {
4.      int i, x, k, flag = 0;
5.      printf("本程序判断合数, 请输入大于 1 的整数, 以 Ctrl+Z 结束\n");
6.      while (scanf("%d", &x) != EOF)
7.      {
    
```

```

8.     flag = 0;
9.     for (i = 2, k = x >> 1; i <= k&&!flag; i++)
10.        if (!(x % i)) {
11.            flag = 1;
12.        }
13.        if (flag == 1) printf("%d 是合数\n", x);
14.        else printf("%d 不是合数\n", x);
15.    }
16.    return 0;
17. }

```



```

C:\Users\muke\source\repos\ x + v
本程序判断合数，请输入大于1的整数，以Ctrl+Z结束
7
7不是合数
9
9是合数
113
113不是合数
15
15是合数

```

图 2-2 程序修改（1）后的程序运行结果示意图

（2）修改实验 2-1 程序，将 for 循环改用 do-while 循环。

解答：

由于 do-while 循环先进行 do 操作再进行判断，循环开始前需先获取一次 x 的值，因此，题目（2）替换后的程序如下所示：

```

1.  #include <stdio.h>
2.  int main()
3.  {
4.      int i, x, k, flag = 0;
5.      printf("本程序判断合数，请输入大于 1 的整数，以 Ctrl+Z 结束\n");
6.      scanf("%d", &x);
7.      do
8.      {
9.          flag = 0;
10.         for (i = 2, k = x >> 1; i <= k; i++)
11.             if (!(x % i)) {
12.                 flag = 1;
13.                 break;
14.             }
15.         if (flag == 1) printf("%d 是合数\n", x);

```

```

16.     else printf("%d 不是合数\n", x);
17. } while (scanf("%d", &x) != EOF);
18. return 0;
19. }
    
```

```

C:\Users\muke\source\repos\
本程序判断合数，请输入大于1的整数，以Ctrl+Z结束
2
2不是合数
3
3不是合数
5
5不是合数
7
7不是合数
4
4是合数
8
8是合数
9
9是合数
    
```

图 2-3 程序修改（2）后的程序运行结果示意图

（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：

根据题意，写出两个辅助函数判断是否为纯粹合数，再遍历三位数，得到答案，因此，题目（3）替换后的程序如下所示：

```

1.  #include <stdio.h>
2.  int isHeshu(int x){//判断是否是合数
3.      int i, k, flag; flag = 0;
4.      for (i = 2, k = x >> 1; i <= k; i++)
5.          if (!(x % i)) {
6.              flag = 1; break; }
7.      if (flag == 1) return 1;
8.      else return 0;
9.  }
10. int PureHeshu(int n){//判断是否是纯粹合数
11.     int ret=0;
12.     if (isHeshu(n))
13.     { n /= 10;
14.       if (isHeshu(n)) {
15.         n /= 10;
    
```

```

16.     if (isHeshu(n)) {
17.         ret = 1; } } }
18.     return ret;}
19. int main(){
20.     for (int m = 100; m < 1000; m++)
21.     { if (PureHeshu(m))
22.         printf("%d ", m); }
23.     return 0;}

```

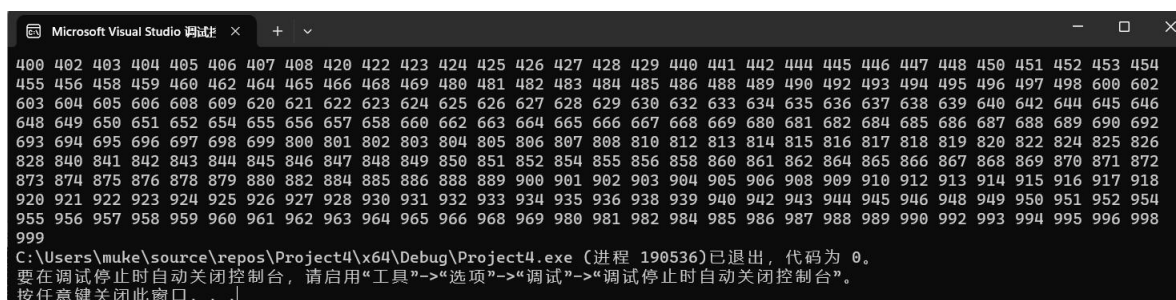


图 2-4 程序修改（3）后的程序运行结果示意图

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； $1000 \leq x < 2000$ ，收取 5% 的税金； $2000 \leq x < 3000$ ，收取 10% 的税金； $3000 \leq x < 4000$ ，收取 15% 的税金； $4000 \leq x < 5000$ ，收取 20% 的税金； $5000 \leq x$ ，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= $1000 \times 5\% + 1000 \times 10\% + 1000 \times 15\% + 501 \times 20\%$ ）。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) 解题思路

1. 输入 x ，根据实际情况， x 声明为双精度浮点数
2. 根据 x 的大小进行选择判断，计算税金
3. 打印结果；
4. 结束

2) 源程序清单

/* if 实现版本*/


```

1.  #include <stdio.h>
2.  int main()
3.  { double x, tax;
4.    printf("请输入工资:");
5.    scanf("%lf", &x);
6.    if (x < 1000) tax = 0;
7.    if (x >= 1000 && x < 2000) tax = (x+1 - 1000) * 0.05;
8.    if (x >= 2000 && x < 3000) tax = 1000*0.05+(x+1-2000) *
    0.1;
9.    if (x >= 3000 && x < 4000) tax = 1000 * 0.05 + 1000*0.1+
    (x+1 - 3000) * 0.15;
10.   if (x >= 4000 && x < 5000) tax = 1000 * 0.05 + 1000 * 0.
    1 + 1000*0.15+(x+1 - 4000) * 0.2;
11.   if (x >= 5000) tax = 1000 * 0.05 + 1000 * 0.1 + 1000 * 0.
    15 +1000*0.2+(x+1 - 5000) * 0.25;
12.   printf("\n 应收税金%lf 元\n", tax);
13. }
    /* switch 实现版本*/

```

```

1.  #include <stdio.h>
2.  int main(){
3.    double x, tax;
4.    printf("请输入工资:");
5.    scanf("%lf", &x);
6.    int a =(int) x / 1000;
7.    switch (a) {
8.      case 0:tax = 0;
9.        break;
10.     case 1:tax = (x + 1 - 1000) * 0.05;
11.       break;
12.     case 2:tax = 1000 * 0.05 + (x + 1 - 2000) * 0.1;
13.       break;
14.     case 3:tax = 1000 * 0.05 + 1000 * 0.1 + (x + 1 - 3000) *
    0.15;
15.       break;
16.     case 4:tax = 1000 * 0.05 + 1000 * 0.1 + 1000 * 0.15 + (x
    + 1 - 4000) * 0.2;
17.       break;
18.     default:tax = 1000 * 0.05 + 1000 * 0.1 + 1000 * 0.15 + 1
    000 * 0.2 + (x + 1 - 5000) * 0.25;
19.       break;
20.    }
21.    printf("\n 应收税金%lf 元\n", tax);
22.    return 0;
23. }

```

3) 测试

(a) 测试数据:

表 2-1 编程题 1 的测试数据

测试用例	程序输入	理论结果
用例 1	1500	25.05
用例 2	2500	100.1
用例 3	4500	400.2

(b) 对应测试数据的运行结果截图



图 2-5 编程题 1 的测试用例 1 的运行结果



图 2-6 编程题 1 的测试用例 2 的运行结果



图 2-7 编程题 1 的测试用例 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) 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) 算法流程如图 2.8 所示。

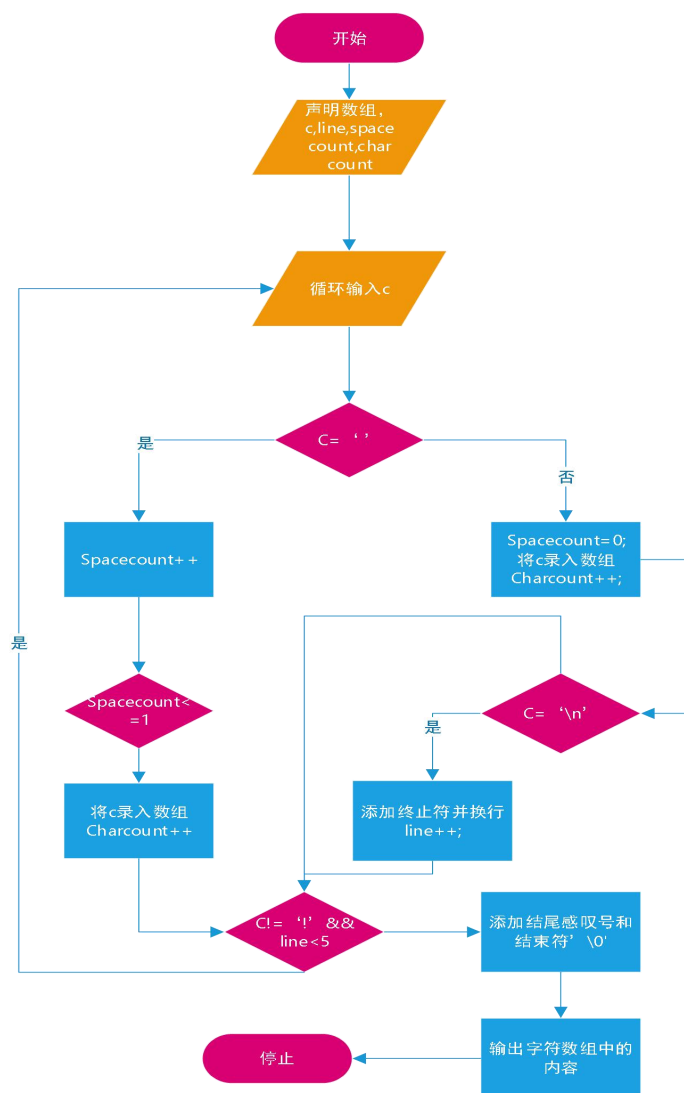


图 2-8 编程题 2 的程序流程图

- 2) 源程序清单

```

1. #include <stdio.h>
2. int main() {
3.     char input[5][51]; char c;

```

```

4.      int line = 0;   int charCount = 0; int spaceCount = 0
      ;
5.      while ((c = getchar()) != '!' && line < 5){
6.          if (c == ' ') {
7.              spaceCount++;           // 空格计数
8.              if (spaceCount <= 1) {   // 保证只有一个空格
9.                  input[line][charCount] = c;
10.                 charCount++;
11.             }
12.         }
13.         else {
14.             spaceCount = 0;           // 归零空格计数
15.             input[line][charCount] = c; // 将 c 存入数组
16.             charCount++;
17.         }
18.         if (c == '\n') {
19.             input[line][charCount] = '\0'; // 添加终止符
并换行
20.             line++;
21.             charCount = 0;
22.             spaceCount = 0;
23.         }
24.     }
25.     input[line][charCount] = '!';     // 添加结尾
26.     input[line][charCount + 1] = '\0';
27.     for (int i = 0; i <= line; i++) {
28.         printf("%s", input[i]);    } // 打印短文
29.     return 0;
30. }

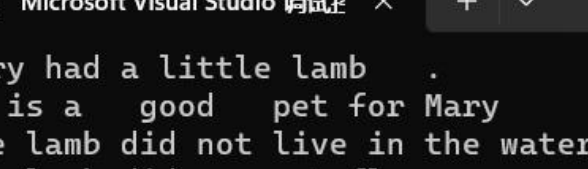
```

3) 测试

(a) 测试数据:

表 2-2 编程题 2 的测试数据

测试输入	理论结果
Mary had a little lamb . It is a good pet for Mary The lamb did not live in the water The lamb did not fly away The lamb liked to look at books!	Mary had a little lamb . It is a good pet for Mary The lamb did not live in the water The lamb did not fly away The lamb liked to look at books!



The screenshot shows a dark-themed window of Microsoft Visual Studio Code. The title bar at the top reads "Microsoft Visual Studio 调试" (Microsoft Visual Studio Debug) with a close button on the right. Below the title bar, there are three icons: a plus sign, a checkmark, and a downward arrow. The main area of the window displays the text of the poem "Mary Had a Little Lamb" in a monospaced font. The text is as follows:

```
Mary had a little lamb .
It is a good pet for Mary
The lamb did not live in the water
The lamb did not fly away
The lamb liked to look at books!
Mary had a little lamb .
It is a good pet for Mary
The lamb did not live in the water
The lamb did not fly away
The lamb liked to look at books!
```

```

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 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

```

$$C_i^0 = 1 \quad (i=0,1,2,\dots)$$

$$C_i^j = C_i^{j-1} * (i - j + 1) / j \quad (j=0,1,2,3,\dots,i)$$

解答：

1) 解题思路

1. 输入整数 N
2. 利用 for 循环打印 N 个空格，并用第二个 for 循环控制每行空格的数量比上一行少两个
3. 利用递推关系求出 coef
4. 利用双层循环打印出杨辉三角
5. 结束

2) 源程序清单

```

1.  #include <stdio.h>
2.  int main() {
3.      int N;
4.      scanf("%d", &N);
5.      for (int i = 0; i <= N; i++) {
6.          for (int a = 1; a <= N; a++) {
7.              printf(" ");
8.          }
9.          for (int space = 1; space <= N - i; space++) {
10.             printf(" ");
11.          }
12.          int coef = 1;
13.          for (int j = 0; j <= i; j++) {
14.              printf("%d", coef);
15.              if (coef < 10)
16.                  printf(" ");
17.              if (coef >= 10)
18.                  printf(" ");
19.              coef = coef * (i - j) / (j+1);
20.          }
21.          printf("\n");
22.      }
23.      return 0;
24.  }
```

3) 测试

(a) 测试数据:

表 2-3 编程题 2 的测试数据

测试输入	理论结果
5	<pre> 1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 1 5 10 10 5 1 </pre>

(b) 对应测试数据的运行结果截图



图 2-10 编程题 3 的测试用例的运行结果

(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) 算法流程如图 2.11 所示。

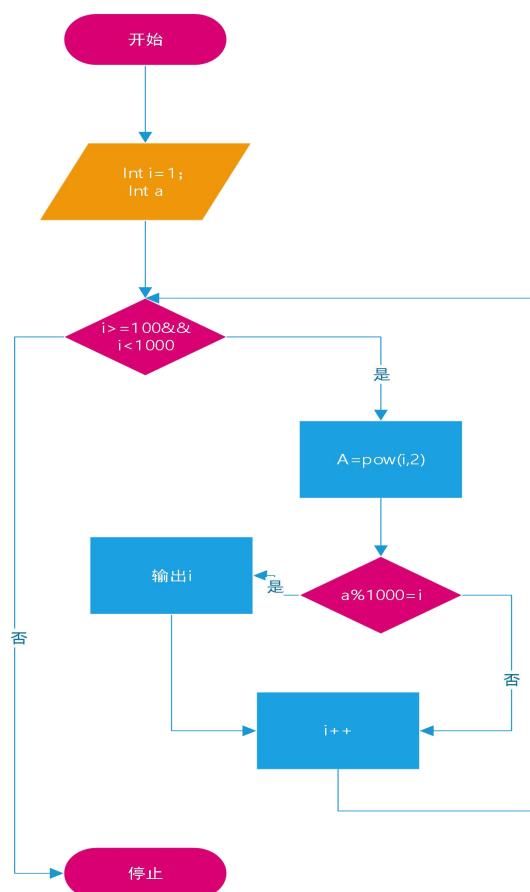


图 2-11 编程题 4 的程序流程图

2) 源程序清单

```

1.  #include <stdio.h>
2.  #include <math.h>
3.  int main()
4.  {
5.      for (int i = 100; i < 1000; i++)
6.      {
7.          int a = pow(i, 2);
8.          if (a % 1000 == i)
9.          {
10.             printf("%d\n", i);
11.          }
12.      }
13.  }
    
```

3) 测试

运行结果如图所示

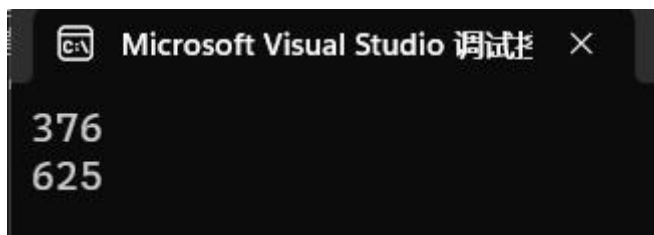


图 2-12 编程题 4 的运行结果

2.3 实验小结

在本实验中，我探索了复合语句、if 语句、switch 语句，以及 for、while、do-while 等循环控制语句的使用，还了解了转移语句和标号语句的应用。此外，我还学习了如何使用集成开发环境（IDE）的调试功能，包括单步执行、设置断点以及观察变量值。在完成这次实验后，我获得了以下几点体会和经验：

1. **善于纠错：**在编写程序时，仍然经常会遇到语法错误，例如拼写错误、缺少分号、括号不匹配等。这些错误可以导致程序无法编译或运行。通过检查代码，查看 IDE 提供的错误消息来识别和纠正语法错误。
2. **重视逻辑，学会调试：**在程序中，尤其是使用循环可能会出现逻辑错误，导致程序的输出不符合预期，或陷入死循环。我们可以使用调试工具，例如设置断点、单步执行，观察程序的执行过程，和中间变量的值，判断错误原因。
3. **善于选择合适结构：**通过实验，我了解每种循环结构和选择结构的特点和用途，知道了 while 和 do-while 的区别，if 和 switch 的区别，学会如何选择适合特定情况的循环结构。
4. **更多样结构的流程图画法：**通过实验，我熟悉 visio 软件的基本使用，知道了如何用流程图表示循环结构和选择结构整理思路。

实验 3 函数与程序结构实验

3.1 实验目的

(1) 熟悉和掌握函数的定义、声明；函数调用与参数传递，函数返回值类型的定义和返回值使用。

(2) 熟悉和掌握不同存储类型变量的使用。

(3) 练习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。

3.2 实验内容

1. 程序改错题

下面是计算 $s=1!+2!+3!+\cdots+n!$ 的源程序($n<20$)。在这个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，使之能够输出如下结果：

k=1 the sum is 1

k=2 the sum is 3

k=3 the sum is 9

.....

k=20 the sum is 2561327494111820313

/*实验 3-1 改错题程序：计算 $s=1!+2!+3!+\cdots+n!$ */

```

1.  #include <stdio.h>
2.  int main(void)
3.  {
4.      int k;
5.      for(k=1;k<=20;k++)
6.          printf("k=%d\tthe sum is %ld\n",k,sum_fac(k));
7.      return 0;
8.  }
9.  long sum_fac(int n)
10. {
11.     long s=0;
12.     int i,fac;
13.     for(i=1;i<=n;i++)

```

```

14.         fac*=i;
15.         s+=fac;
16.         return s;
17.     }

```

解答：

(1) 错误修改：

1) 第 3 行未声明 sum-fac 函数，正确形式为：

```
long long sum_fac(int n);
```

2) 第 9 行的函数返回值类型出错，正确形式为：

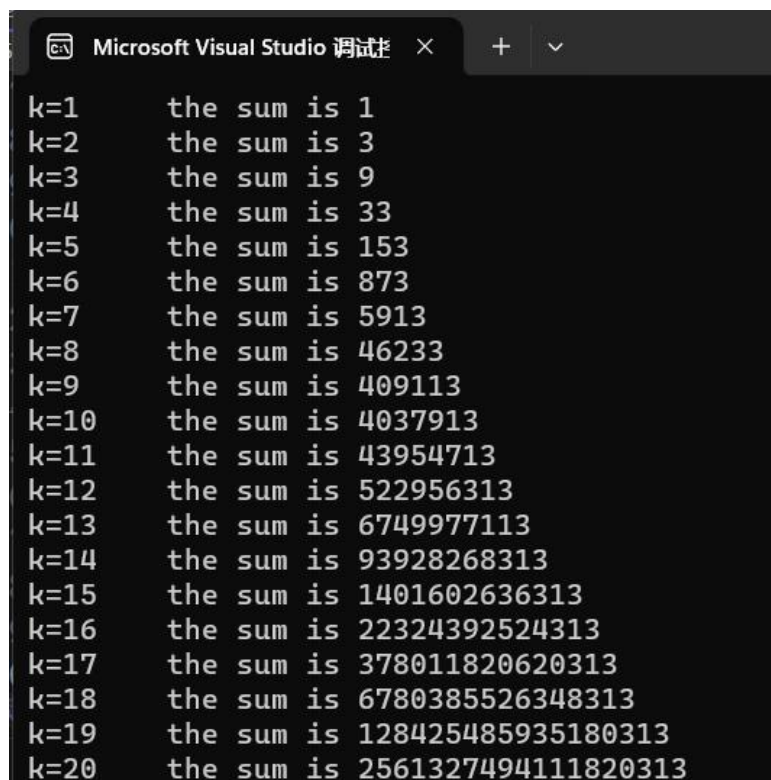
```
long long sum_fac(int n);
```

3) 第 11 行 s 的数据类型出错，正确形式为：long long s = 0;

4) 第 12 行 fac 的数据类型出错且未初始化，正确形式为：

```
long long fac = 1;
```

(2) 错误修改后运行结果：



```

k=1    the sum is 1
k=2    the sum is 3
k=3    the sum is 9
k=4    the sum is 33
k=5    the sum is 153
k=6    the sum is 873
k=7    the sum is 5913
k=8    the sum is 46233
k=9    the sum is 409113
k=10   the sum is 4037913
k=11   the sum is 43954713
k=12   the sum is 522956313
k=13   the sum is 6749977113
k=14   the sum is 93928268313
k=15   the sum is 1401602636313
k=16   the sum is 22324392524313
k=17   the sum is 378011820620313
k=18   the sum is 6780385526348313
k=19   the sum is 128425485935180313
k=20   the sum is 2561327494111820313

```

图 3-1 错误修改后的程序运行结果示意图

2. 程序修改替换题

(1) 根据 $\sum_{i=1}^n i! = \sum_{i=1}^{n-1} i! + n!$ 将实验 3-1 改错题程序中 `sum_fac` 函数修改为一个

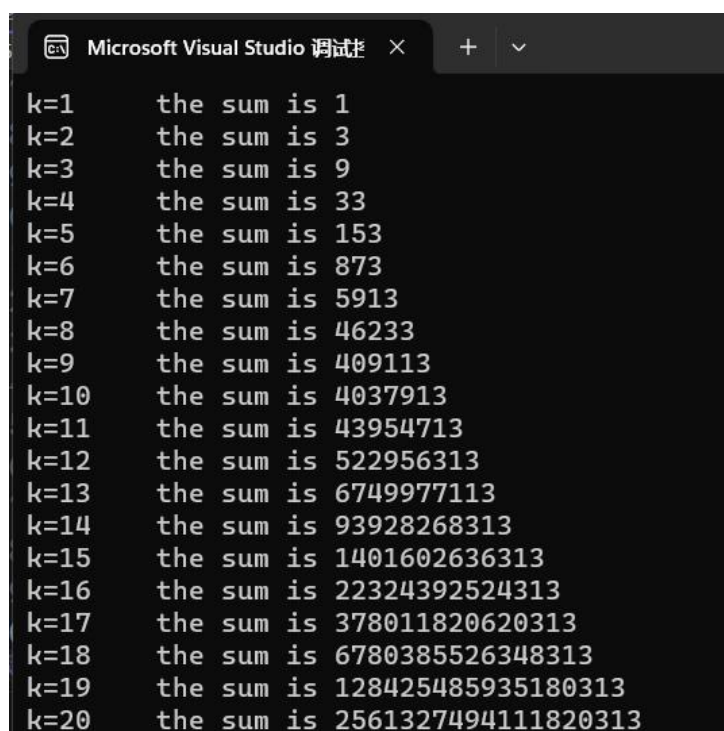
递归函数，用递归的方式计算 $\sum_{i=1}^n i!$ 。

解答：

替换后的程序如下所示：

```

1.  #include <stdio.h>
2.  int main(void)
3.  {
4.      long long sum_fac(int n);
5.      int k;
6.      for (k = 1; k <= 20; k++)
7.          printf("k=%d\tthe sum is %lld\n", k, sum_fac(k));
8.      return 0;
9.  }
10. long long sum_fac(int n)
11. {
12.     long long s = 0;
13.     long long i, fac = 1;
14.     for (i = 1; i <= n; i++) {
15.         fac *= i;
16.     }
17.     s += fac;
18.     if (n == 1)
19.         return s;
20.     else
21.         return s + sum_fac(n - 1);
22. }
```



```

k=1    the sum is 1
k=2    the sum is 3
k=3    the sum is 9
k=4    the sum is 33
k=5    the sum is 153
k=6    the sum is 873
k=7    the sum is 5913
k=8    the sum is 46233
k=9    the sum is 409113
k=10   the sum is 4037913
k=11   the sum is 43954713
k=12   the sum is 522956313
k=13   the sum is 6749977113
k=14   the sum is 93928268313
k=15   the sum is 1401602636313
k=16   the sum is 22324392524313
k=17   the sum is 378011820620313
k=18   the sum is 6780385526348313
k=19   the sum is 128425485935180313
k=20   the sum is 2561327494111820313
    
```

图 3-2 程序修改替换（1）后的程序运行结果示意图

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) 下面是计算 $s = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$ 的源程序，其中 x 是浮点数，

n 是整数。从键盘输入 x 和 n，然后计算 s 的值。修改该程序中的 sum 和 fac 函数，使之计算量最小。

/*实验 3-2 程序修改替换第(2)题程序：根据公式计算 s*/

```

1.  #include<stdio.h>
2.  double mulx(double x,int n);
3.  long fac(int n);
4.  double sum(double x,int n)
5.  {
6.      int i;
7.      double z=1.0;
8.      for(i=1;i<=n;i++)
9.      {
10.         z=z+mulx(x,i)/fac(i);
11.      }
12.      return z;
13.  }
14.  double mulx(double x,int n)
15.  {
16.      int i;
    
```

```

15. double z=1.0;
16. for(i=0;i<n;i++)
17. {
18.     z=z*x;
19. }
20. return z;
21. }
22. long fac(int n)
23. {
24.     int i;
25.     long h=1;
26.     for(i=2;i<=n;i++)
27.     {
28.         h=h*i;
29.     }
30.     return h;
31. }
32. int main()
33. {
34.     double x;
35.     int n;
36.     printf("Input x and n:");
37.     scanf("%lf%d",&x,&n);
38.     printf("The result is %lf:",sum(x,n));
39.     return 0;
40. }

```

解答:

采用静态数据类型优化, 修改后的程序如下所示:

```

1. #include <stdio.h>
2. /* 计算  $s=1+x+x^2/2!+x^3/3!$  */
3. double mulx(double x)
4. {
5.     static double z = 1.0;
6.     z *= x;
7.     return z;
8. }
9. long fac(int n)
10. {
11.     static long f = 1;
12.     //静态局部变量, 静态局部变量使用 static 修饰符定义,
13.     //即使在声明时未赋初值, 编译器也会把它初始化为 0。且静态局部变量
        存储于进程的
14.     //全局数据区, 即使函数返回, 它的值也会保持不变。

```

```

15.     f *= n;
16.     return f;
17. }
18. double sum(double x, int n)
19. {
20.     int i;
21.     double z = 1.0;
22.     for (i = 1; i <= n; i++)
23.     {
24.         z = z + mulx(x) / fac(i);
25.     }
26.     return z;
27. }
28. int main() {
29.     double x;
30.     int n;
31.     printf("Input x and n:");
32.     scanf("%lf%d", &x, &n);
33.     printf("The result is %lf:", sum(x, n));
34.     return 0;
35. }
    
```

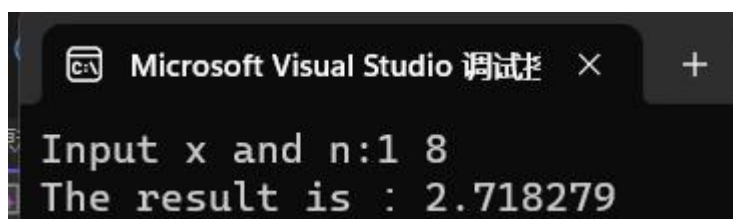


图 3-3 程序修改替换（2）后的程序运行结果示意图

根据数学拟合函数，结果应与 e 接近，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. 跟踪调试题

下面是计算 fibonacci 数列前 n 项和的源程序，现要求单步执行该程序，在 watch 窗口中观察 lk, sum, n 值。具体操作如下：

- （1）设输入 5，观察刚执行完“scanf(“%d”,&k);”语句时， sum 、 k 的值是多少？
- （2）在从 main 函数第一次进入 fibonacci 函数前的一刻，观察各变量的值是多少？返回后光条停留在哪个语句上？
- （3）在从 main 函数第一次进入 fibonacci 函数后的一刻，观察光条从 main 函数

“sum+=fabonacci(i);”语句调到了哪里？

(4) 在 fabonacci 函数内部单步执行，观察函数的递归执行过程。体会递归方式实现的计算过程是如何完成数计算的，并特别注意什么时刻结束递归，然后直接从第一个 return 语句返回到了哪里？

(5) 在 fabonacci 函数递归执行过程中观察参数 n 的变化情况，并回答为什么 k、sum 在 fabonacci 函数内部不可见？

/*实验 3-3 跟踪调试题程序：计算 fabonacci 数列前 n 项和*/

```

1.  #include<stdio.h>
2.  int main(void)
3.  {
4.      int i,k;
5.      long sum=0,fabonacci(int n);
6.      printf("Inut n:");
7.      scanf("%d",&k);
8.      for(i=1;i<=k;i++){
9.          sum+=fabonacci(i);
10.         printf("i=%d\tthe sum is %ld\n",i,sum);
11.     }
12.     return 0;
13. }
14. long fabonacci(int n)
15. {
16.     if(n==1 || n==2)
17.         return 1;
18.     else
19.         return fabonacci(n-1)+fabonacci(n-2);
20. }
```

解答：

(1) sum,k 的值分别是 0 和 5。

名称	值	类型
sum	0	long
k	5	int

图 3-4 跟踪调试题（1）的程序运行截图

(2) 如图，sum,k,i 的值分别为 0，5，1，光条在 sum+=fabonacci(i);语句前。

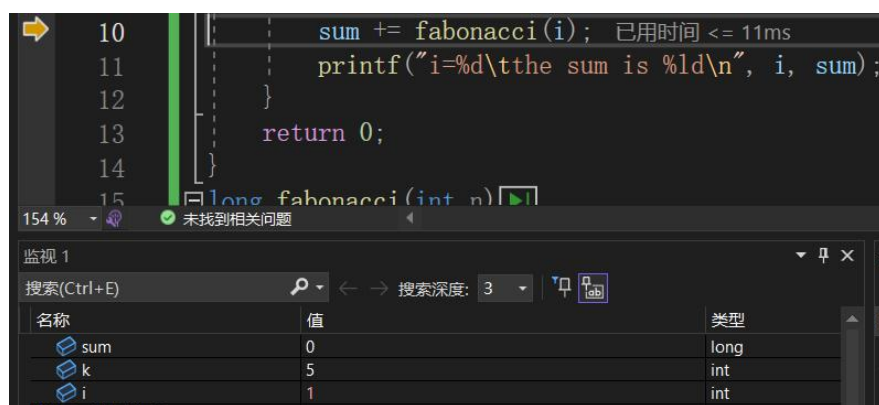


图 3-5 跟踪调试题（2）的程序运行截图

(3) 光条调到了 fibonacci 函数体的开始,如图所示:

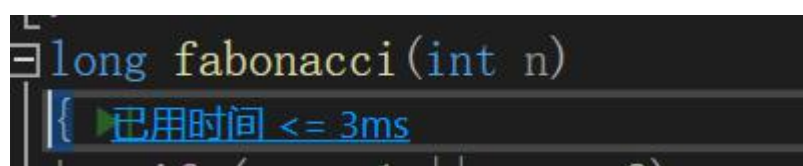


图 3-6 跟踪调试题（3）的程序运行截图

(4) 观察递归可发现, 当 $n=1$ 或 2 时, 函数直接返回 1 , 当 $n \geq 3$ 时, 函数调用自身两次, 分别计算 $\text{fibonacci}(n-1)$ 和 $\text{fibonacci}(n-2)$, 若 $\text{fibonacci}(n-1)$ 和 $\text{fibonacci}(n-2)$ 中 $n-1$ 或 $n-2 \geq 3$, 函数会继续调用自身, 直到 $n=1$ 或 2 。当结束时, 光条回到 $\text{sum} += \text{fibonacci}(i);$ 处, 如图:

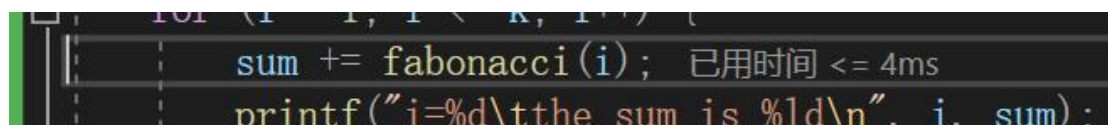


图 3-7 跟踪调试题（4）的程序运行截图

(5) 设输入 3 , n 的值经历了从 $1-2-3-2-3-1-3$ 的转变, 符合上一问中所解释的递归调用过程。定义在函数内部的变量称为局部变量 (Local Variable), 它的作用域仅限于函数内部, 离开该函数后就是无效的, k, sum 定义在 main 函数内, 只在 main 函数内部可见。fibonacci 函数只能访问自己的参数 n , 而不能访问 main 函数中的局部变量 k 和 sum 。

名称	值	类型
sum	0	long
k	5	int
i	1	int
n	1	int

图 3-8 跟踪调试题（5）的程序运行截图

4. 程序设计

(1) 编程验证歌德巴赫猜想：任何一个大于等于 4 的偶数都是两个素数之和。要求设计一个函数，接受形参 n ，以 “ $n=n_1+n_2$ ” 的形式输出结果，若有多种分解情况，取 n_1 最小的一个输出。

例如： $n=6$ ，输出 “ $6=3+3$ ”。

main 函数循环接收从键盘输入的整数 n ，如果 n 是大于或等于 4 的偶数，调用上述函数进行验证。

解答：

1) 算法流程如图 3.9 所示。

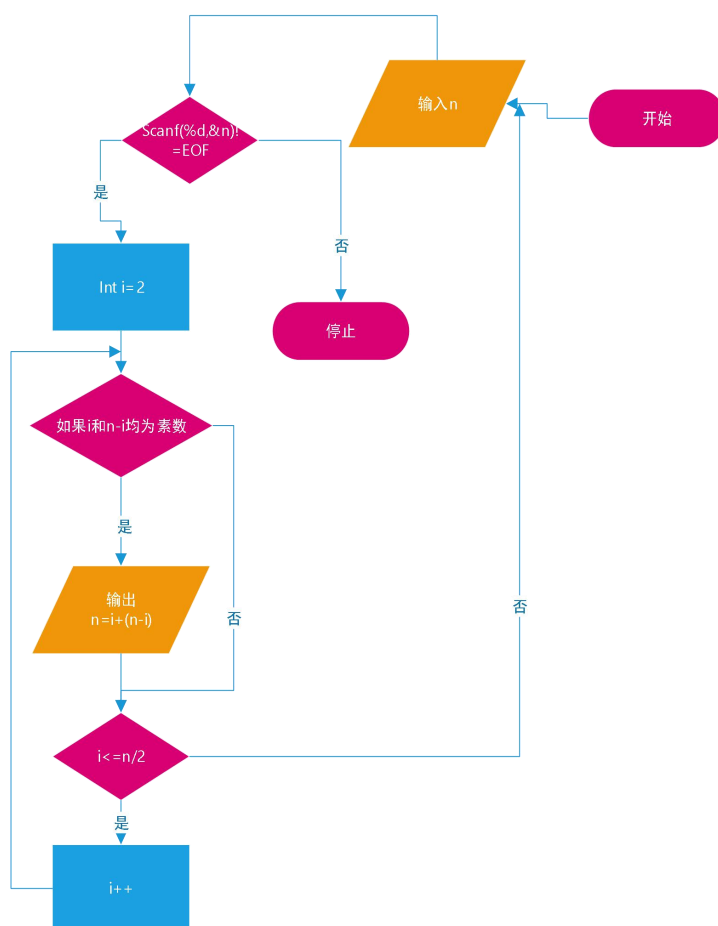


图 3-9 编程题 1 的程序流程图

2) 源程序清单

```

1.  #include<stdio.h>
2.  #include<math.h>
3.  int isPrime(int x)//素数
4.  {
5.      int ret = 1;
6.      int i;
7.      if (x == 1 || (x % 2 == 0 && x != 2))
8.          ret = 0;
9.      for (i = 3; i < sqrt(x); i += 2)
10.     {
11.         if (x % i == 0)
12.         {
13.             ret = 0;
14.             break;
15.         }
16.     }
17.     if (ret == 1)
18.         return ret;
19. }
20. void gedebahe(int n)//哥德巴赫猜想的函数和输出
21. {
22.
23.     for (int i = 2; i <= n / 2; i++)
24.     {
25.         if (isPrime(i) && isPrime(n - i))
26.         {
27.             printf("%d=%d+%d\n", n, i, n - i);
28.             return;
29.         }
30.     }
31. }
32.
33. int main()
34. {
35.     int n;
36.     while (scanf("%d", &n) != EOF)
37.     {
38.         gedebahe(n);
39.     }
40.     return 0;
41. }

```

3) 测试

(a) 测试数据:

表 3-1 编程题 1 的测试数据

测试用例	程序输入	理论结果
用例 1	48	$48=5+43$
用例 2	90	$90=7+83$
用例 3	10	$10=3+7$

(b) 对应测试数据的运行结果截图

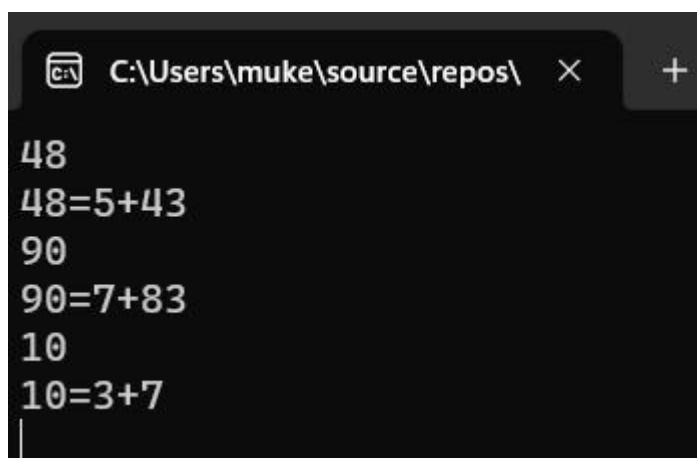


图 3-10 编程题 1 的测试用例的运行结果

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) 完全数 (Perfect number)，又称完美数或完备数，特点是它的所有真因子（即除了自身以外的约数，包括 1）之和恰好等于它本身。例如 $6=1+2+3$ ， $28=1+2+4+7+14$ 等。

编程寻找 10000 以内的所有完全数。

要求设计一个函数，判定形参 n 是否为完全数，如果是，返回 1，否则返回 0。在 main 函数中调用该函数求 10000 以内的所有完全数，并以完全数的真因子之和的形式输出结果，例如 “ $6=1+2+3$ ”。程序输出中，每个完全数单独占一行。
解答：

1) 算法流程如图 3.11 所示。

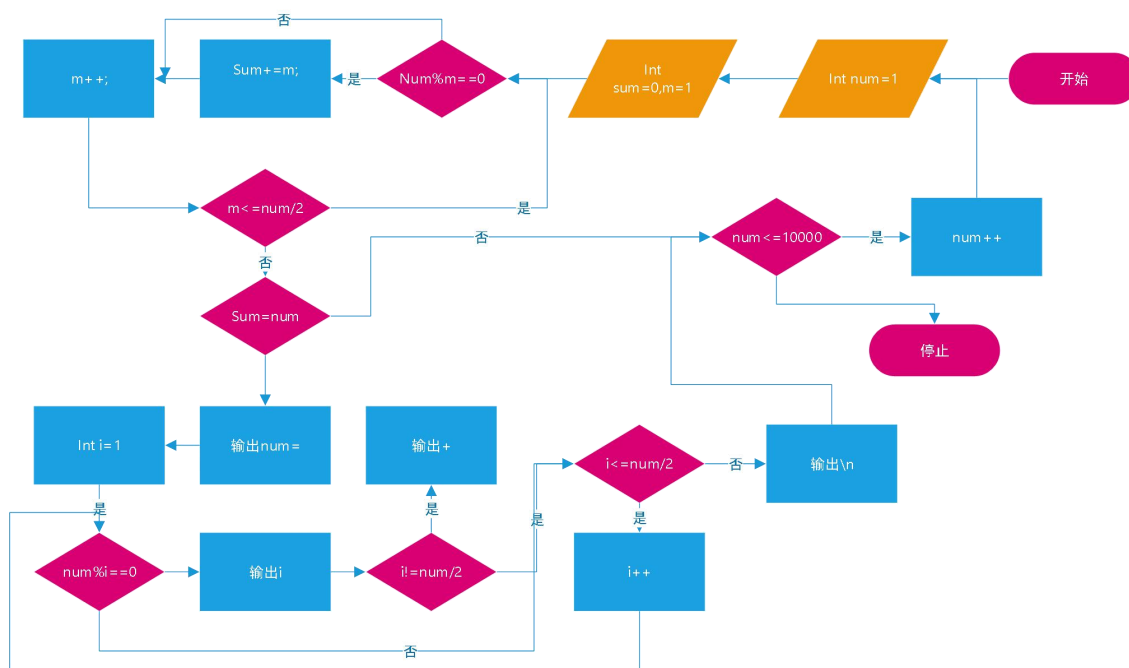


图 3-11 编程题 2 的程序流程图

2) 源程序清单

```

1.  #include <stdio.h>
2.  int wanshu(int n)
3.  {
4.      int sum = 0;
5.      for (int i = 1; i <= n / 2; i++)
6.      {
7.          if (n % i == 0) { //找到一个因子
8.              sum += i;
9.          }
10.     }
11.     if (sum == n) //如果两者相等
12.     {
13.         return 1;
14.     }
15.     else
16.     {
17.         return 0;
18.     }
19. }
20.
21. int main() {
22.     for (int num = 1; num <= 10000; num++)
    
```

```

23.     {
24.         if (wanshu(num))
25.         {
26.             printf("%d=", num);
27.             for (int i = 1; i <= num / 2; i++) {
28.                 if (num % i == 0) {
29.                     printf("%d", i);
30.                     if (i != num / 2) {
31.                         printf("+");
32.                     }
33.                 }
34.             }
35.             printf("\n");
36.         }
37.     }
38.     return 0;
39. }
    
```

3) 测试

对应测试数据的运行结果截图

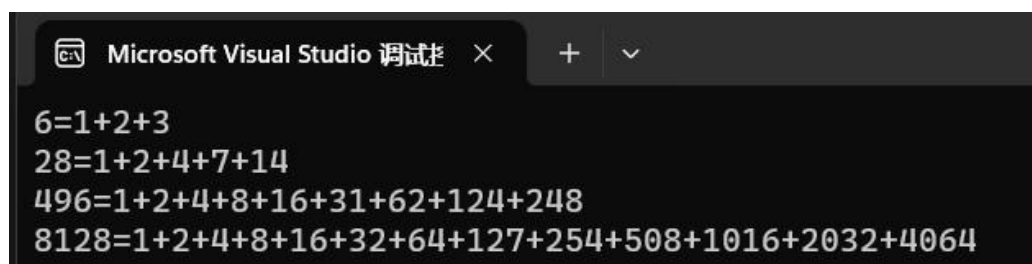


图 3-12 编程题 2 的运行结果

(3) 自幂数是指一个 n 位数，它的每个位上的数字的 n 次幂之和等于它本身。水仙花数是 3 位的自幂数，除此之外，还有 4 位的四叶玫瑰数、5 位的五角星数、6 位的六合数、7 位的北斗星数、8 位的八仙数等。

编写一个函数，判断其参数 n 是否为自幂数，如果是，返回 1；否则，返回 0。要求 main 函数能反复接收从键盘输入的整数 k ， k 代表位数，然后调用上述函数求 k 位的自幂数，输出所有 k 位自幂数，并输出相应的信息，例如“3 位的水仙花数共有 4 个 153, 370, 371, 407”。当 $k=0$ 时程序结束执行。

解答：

1) 算法流程如图 3.13 所示。

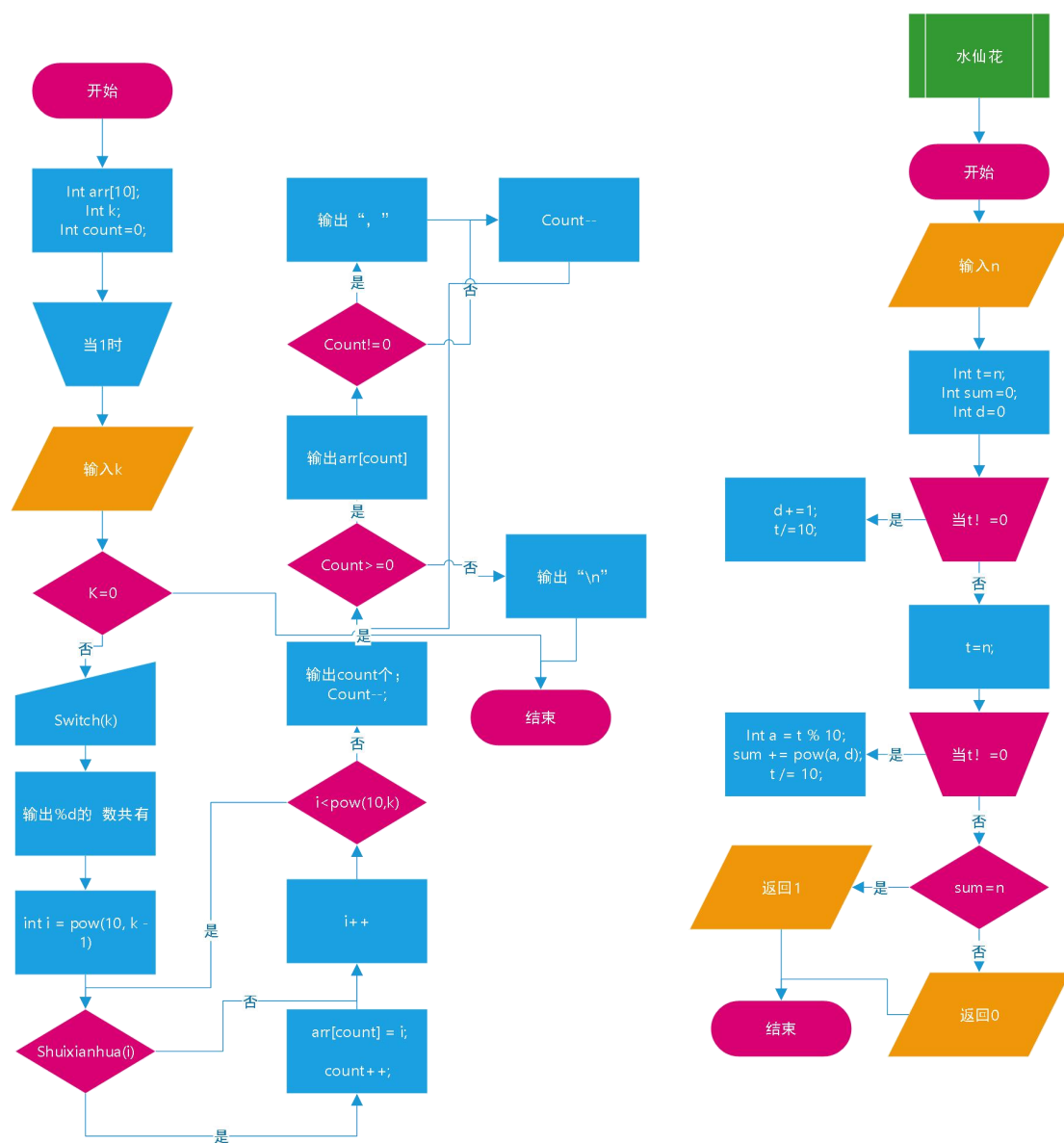


图 3-13 编程题 3 的程序流程图

2) 源程序清单

```

1. #include <stdio.h>
2. #include <math.h>
3. int shuixianhua(int n) {
4.     int t = n;
5.     int sum = 0;
6.     int d = 0;
7.     while (t != 0)
8.     {
9.         d++;
    
```

```

10.         t /= 10;
11.     }
12.     t = n;
13.     while (t != 0)
14.     {
15.         int a = t % 10;
16.         sum += pow(a, d);
17.         t /= 10;
18.     }
19.     if (sum == n)
20.         return 1;
21.     else
22.         return 0;
23. }
24. int main()
25. {
26.     int arr[10];
27.     int k;
28.     while (1)
29.     {
30.         scanf("%d", &k);
31.         if (k == 0) {
32.             break;
33.         }
34.         int count = 0;
35.         switch (k)
36.         {
37.             case 3:
38.                 printf("%d 位的水仙花数共有", k);
39.                 break;
40.             case 4:
41.                 printf("%d 位的四叶玫瑰数共有", k);
42.                 break;
43.             case 5:
44.                 printf("%d 位的五角星数共有", k);
45.                 break;
46.             case 6:
47.                 printf("%d 位的六合数共有", k);
48.                 break;
49.             case 7:
50.                 printf("%d 位的北斗星数共有", k);
51.                 break;
52.             case 8:
53.                 printf("%d 位的八仙数共有", k);

```



```

54.         break;
55.     default:
56.         printf("%d 位的数共有", k);
57.         break;
58.     }
59.     for (int i = pow(10, k - 1); i < pow(10, k); i++)
60.     {
61.         if (shuixianhua(i))
62.         {
63.             arr[count] = i;
64.             count++;
65.         }
66.     }
67.     printf("%d 个", count);
68.     for (count = count - 1; count >= 0; count--)
69.     {
70.         printf("%d", arr[count]);
71.         if (count != 0)
72.         {
73.             printf(",");
74.         }
75.     }
76.     printf("\n");
77.     return 0;
78. }
79.
80. }

```

3) 测试

(a) 测试数据:

表 3-2 编程题 3 的测试数据

测试用例	程序输入	理论结果
用例 1	3	3 位的水仙花数共有 4 个 407,371,370,153
用例 2	4	4 位的四叶玫瑰数共有 3 个 9474,8208,1634
用例 3	5	5 位的五角星数共有 3 个 93084,92727,54748

(b) 对应测试数据的运行结果截图

```
3
3位的水仙花数共有4个407,371,370,153
4
4位的四叶玫瑰数共有3个9474,8208,1634
5
5位的五角星数共有3个93084,92727,54748
6
6位的六合数共有1个548834
7
7位的北斗星数共有4个9926315,9800817,4210818,1741725
```

图 3-14 编程题 3 的测试用例的运行结果

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。

3.3 实验小结

在这次上机实验中，我们旨在熟悉和掌握函数的定义、声明，函数调用与参数传递，函数返回值类型的定义和返回值使用，同时了解不了同存储类型变量的使用。此外，我们练习了使用集成开发环境的调试功能，包括单步执行、设置断点、观察变量值，并借此观察函数递归的过程，以下是我在本次实验中遇到的问题和收获。

1. **熟悉了函数的使用：**通过实验，我学习了函数的定义，声明，返回值类型和使用，在跟踪调试过程中，我了解的递归的过程，并知道了其缺点，通过调试功能观察到变量在函数调用，递归过程中的变化，对计算机有了更深的了解。
2. **纠错能力得到提升：**在改错题中，我意识到数据类型错误，却忘记了 `printf` 输出中格式符的变化，通过实验，锻炼了我思维的周密性。
3. **重视数学与编程相结合：**本次实验的程序设计题均为数学相关问题，在实验过程中，经常不知道如何确定循环的终止条件，这时需要借助数学只是判断，如完数一定是偶数，可确定 `for` 循环中 $i \leq n/2$ 。
4. **了解了算法的优化：**通过程序修改替换题，我学会了利用静态数据类型优化算法，减少计算量。在水仙花数题中，通过摸索我知道了可以用数组储存后面需要用到的变量，避免了重复的运算。本次实验对算法思维有一定帮助。

实验 4 编译预处理实验

4.1 实验目的

- (1) 掌握文件包含、宏定义、条件编译和 `assert` 宏的使用；
- (2) 练习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。
- (3) 熟悉多文件编译技术

4.2 实验内容

1. 程序改错

下面是用宏来计算平方差、交换两数的源程序.在这个源程序中存在若干错误，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 4-1 改错与跟踪调试题程序：计算平方差、将换两数*/

```
1.  #include<stdio.h>
2.  #define SUM a+b
3.  #define DIF a-b
4.  #define SWAP(a,b)  a=b,b=a
5.  int main()
6.  {
7.      int a,b;
8.      printf("Input two integers a, b:");
9.      scanf("%d%d", &a,&b);
10.     printf("\nSUM=%d\n the difference between square of a
    and square of b is:%d",SUM, SUM*DIF);
11.     SWAP(a,b);
12.     printf("\nNow a=%d,b=%d\n",a,b);
13.     return 0;
14. }
```

解答：

(1) 错误修改：

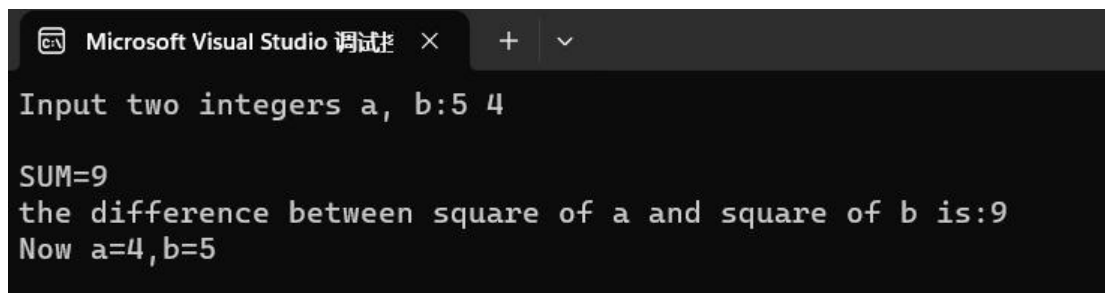
- 1) 第 2 行宏定义有误，正确形式为：`#define SUM (a+b)`

2)第 3 行宏定义有误，正确形式为：`#define DIF (a-b)`

3)第 4 行宏定义有误，正确形式为：

`#define SWAP(a,b) a=a*b,b=a/b,a=a/b`

(2) 错误修改后运行结果：



```
Microsoft Visual Studio 调试
Input two integers a, b:5 4
SUM=9
the difference between square of a and square of b is:9
Now a=4,b=5
```

图 4-1 错误修改后的程序运行结果示意图

2. 程序修改替换

下面是用函数实现求三个数中最大数、计算两浮点数之和的程序。在这个源程序中存在若干语法和逻辑错误。

要求：（1）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；

（2）用带参数的宏替换函数 `max`，来实现求最大数的功能。

/*实验 4-2 程序修改替换题程序*/

```
1.  #include<stdio.h>
2.  int main(void)
3.  {
4.      int a, b, c;
5.      float d, e;
6.      printf("Input three integers:");
7.      scanf("%d %d %d",&a,&b,&c);
8.      printf("\nThe maximum of them is %d\n",max(a,b,c));
9.
10.     printf("Input two floating point numbers:");
11.     scanf("%f %f",&d,&e);
12.     printf("\nThe sum of them is  %f\n",sum(d,e));
13.     return 0;
14. }
15.
16. int max(int x, int y, int z)
17. {
```

```

18.  int m=z;
19.  if (x>y)
20.      if(x>z) m=x;
21.  else
22.      if(y>z) m=y;
23.      return m;
24.  }
25.
26.  float sum(float x, float y)
27.  {
28.      return x+y;
29.  }

```

解答：

源程序未声明函数，可用宏定义替换

```

1.  int max(int x, int y, int z);
2.  float sum(float x, float y);

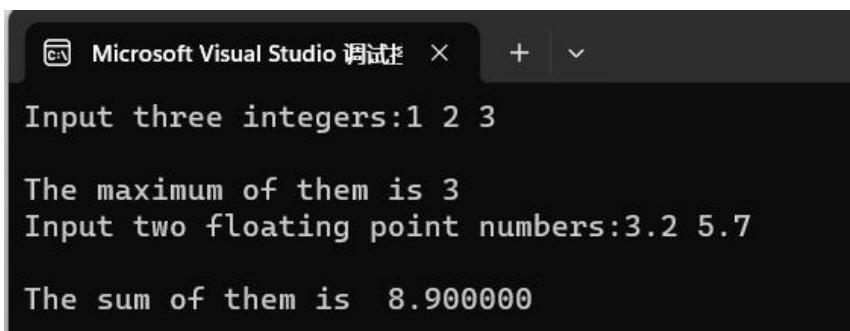
```

替换后的程序如下所示：

```

1.  #include<stdio.h>
2.  #define max(a,b,c) ( a > b ) ? ( a > c ? a : c ) : ( c >
    b ? c : b )
3.  float sum(float x, float y)
4.  {
5.      return x + y;
6.  }
7.  int main(void)
8.  {
9.
10.     int a, b, c;
11.     float d, e;
12.     printf("Input three integers:");
13.     scanf("%d %d %d", &a, &b, &c);
14.     printf("\nThe maximum of them is %d\n", max(a, b, c));
15.
16.     printf("Input two floating point numbers:");
17.     scanf("%f %f", &d, &e);
18.     printf("\nThe sum of them is  %f\n", sum(d, e));
19.     return 0;
20. }

```



```

Microsoft Visual Studio 调试
Input three integers:1 2 3
The maximum of them is 3
Input two floating point numbers:3.2 5.7
The sum of them is 8.900000
    
```

图 4-2 程序修改替换后的程序运行结果示意图

3. 跟踪调试

下面程序利用 R 计算圆的面积 s，以及面积 s 的整数部分。现要求：

- (1) 修改程序，使程序编译通过且能运行；
- (2) 单步执行。进入函数 `integerl_fraction` 时，watch 窗口中 x 为何值？在返回 main 时，watch 窗口中 i 为何值？
- (3) 修改程序，使程序能输出面积 s 值的整数部分（要求四舍五入），不会输出错误信息 `assertion failed`。

/*实验 4-3 跟踪调试题程序利用 R 计算圆的面积 s*/

```

1.  #define R
2.  int main(void)
3.  {
4.      float r, s;
5.      int s_integer=0;
6.      printf("Input a number: ");
7.      scanf("%f",&r);
8.      #ifdef R
9.          s=3.14159*r*r;
10.         printf("Area of round is: %f\n",s);
11.         s_integer=integer_fraction(s);
12.         assert((s-s_integer)<0.5);
13.         printf("The integer fraction of area is %d\n", s_i
            nteger);
14.     #endif
15.     return 0;
16. }
17.
18. int integer_fraction(float x)
    
```

```
19. {
20.     int i=x;
21.     return i;
22. }
```

解答:

(1) 添加头文件和函数的声明

```
#include<stdio.h>
```

```
#include <assert.h>
```

```
int integer_fraction(float x);
```

(2)x 的值为 3.14159012,i 为 3

名称	值	类型
x	3.14159012	float
i	3	int

图 4-3 跟踪调试题的单步执行图

(3) 修改后的程序如下所示:

```
1.  #include<stdio.h>
2.  #include <assert.h>
3.  #include<math.h>
4.  #define R
5.  int main(void)
6.  {
7.      float r, s;
8.      int s_integer = 0;
9.      printf("Input a number: ");
10.     scanf("%f", &r);
11.     #ifdef R
12.         s = 3.14159 * r * r;
13.         printf("Area of round is: %f\n", s);
14.         s_integer = round(s); //取整
15.         assert((s - s_integer) < 1.0);
16.         printf("The integer fraction of area is %d\n", s_inte
            ger);
17.     #endif
18.     return 0;
19. }
```

修改后的运行结果:

```
Input a number: 2
Area of round is: 12.566360
The integer fraction of area is 13
```

图 4-4 跟踪调试题的运行结果图

4. 程序设计

(1) 三角形的面积是 $area = \sqrt{s(s-a)(s-b)(s-c)}$ ，其中 $s = (a+b+c)/2$ ， a, b, c 为三角形的三边，要求编写程序用带参数的宏来计算三角形的面积。定义两个带参数的宏，一个用来求 s ，另一个用来求 $area$ 。

解答：

1) 算法流程如图 4.5 所示。

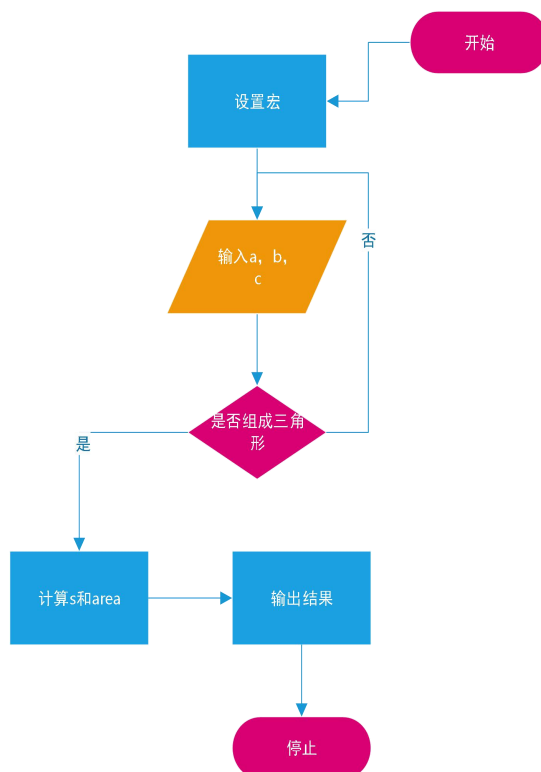


图 4-5 编程题 1 的程序流程图

2) 程序清单

```
1. #include <stdio.h>
2. #include <math.h>
3. #define s ( ( a + b + c ) / 2 )
```



```

4.  #define area ( sqrt( s * ( s - a ) * ( s - b ) * ( s - c )
    ) )
5.
6.  double a, b, c;
7.  int main()
8.  {
9.  input:
10.     printf("请输入三条边长度:");
11.     scanf("%lf%lf%lf", &a, &b, &c);
12.     if (a + b <= c || a + c <= b || b + c <= a)
13.     {
14.         printf("不是三角形, 请重新输入\n");
15.         goto input;
16.     }
17.     else {
18.         printf("%lf", area);
19.     }
20.     return 0;
21. }

```

3) 测试

(a) 测试数据:

表 4-1 编程题 1 的测试数据

测试用例	程序输入	理论结果
用例 1	3, 4, 5	6
用例 2	2, 2, 2	1.732
用例 3	1, 2, 3	错误, 重新输入

b) 对应测试数据的运行结果截图

```

请输入三条边长度:3 4 5
6.000000
请输入三条边长度:2 2 2
1.732051
请输入三条边长度:1 2 3
不是三角形, 请重新输入

```

图 4-6 编程题 1 的测试用例的运行结果

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) 用条件编译方法来编写程序。输入一行英文字符序列，可以任选两种方式之一输出：一为原文输出；二为变换字母的大小写后输出。例如小写‘a’变成大写‘A’，大写‘D’变成小写‘d’，其他字符不变。用#define 命令控制是否变换字母的大小写。例如，#define CHANGE 1 则输出变换后的文字，若#define CHANGE 0 则原文输出。

解答：

1) 算法流程如图 1.1 所示。

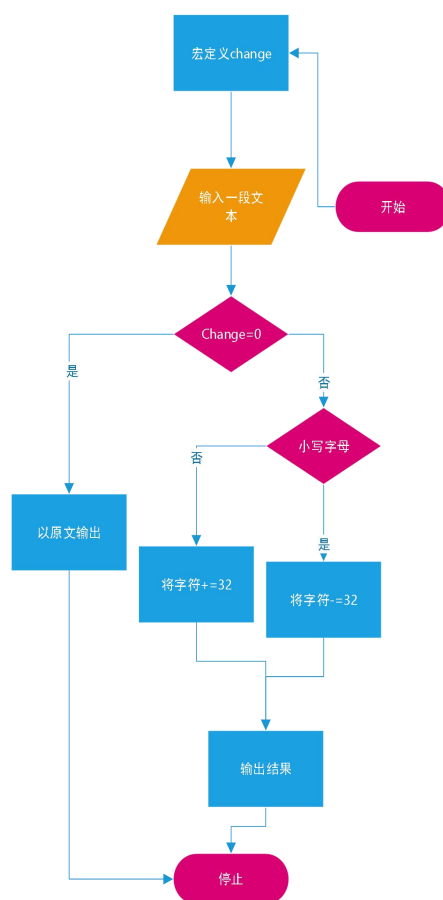


图 4-7 编程题 2 的程序流程图

2) 源程序清单

```

1.  #include <stdio.h>
2.  #include <string.h>
3.  #define change 1
4.
5.  char s[100];
6.  int main()
    
```

```
7.  {
8.    fgets(s, 100, stdin);
9.    #if change
10.    fputs(s, stdout);
11.    #endif
12.
13.    #if !change
14.    int count = strlen(s);
15.    for (int i = 0; i < count; i++)
16.    {
17.        if (s[i] >= 'a' && s[i] <= 'z')
18.        {
19.            s[i] -= 32;
20.        }
21.        else if (s[i] >= 'A' && s[i] <= 'Z')
22.        {
23.            s[i] += 32;
24.        }
25.    }
26.    fputs(s, stdout);
27.    #endif
28.    return 0;
29. }
```

3) 测试

(a) 测试数据:

表 4-2 编程题 2 的测试数据

测试用例	程序输入	理论结果
用例 1	abcdEFG(change 为 0)	abcdEFG
用例 2	abcdEFG(change 为 1)	ABCDefg

(b) 对应测试数据的运行结果截图



图 4-8 编程题 2 的测试用例 1 的运行结果

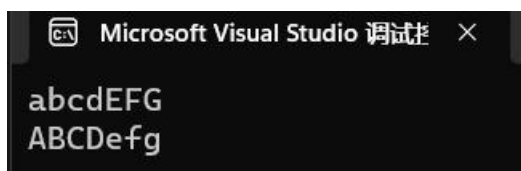


图 4-9 编程题 2 的测试用例 2 的运行结果

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) 假设一个 C 程序由 file1.c 和 file2.c 两个源文件及一个 file.h 头文件组成，file1.c、file2.c 和 file.h 的内容分别如下所述。试编辑该多文件 C 程序，补充 file.h 头文件内容，然后编译和链接。然后运行最后生成的可执行文件。

/*源文件 file1.c 的内容*/

```
1.  #include "file.h"
2.  int x,y;          /* 外部变量的定义性说明 */
3.  char ch;          /* 外部变量的定义性说明 */
4.  int main(void)
5.  {
6.      x=10;
7.      y=20;
8.      ch=getchar();
9.      printf("in file1 x=%d,y=%d,ch is %c\n",x,y,ch);
10.     func1();
11.     return 0;
12. }
```

/*源文件 file2.c 的内容为: */

```
1.  #include "file.h"
2.  void func1(void)
3.  {
4.      x++;
5.      y++;
6.      ch++;
7.      printf("in file2 x=%d,y=%d,ch is %c\n",x,y,ch);
8.  }
```

解答:

1) 解题思路:

在文件中包含 stdio.h 头文件,声明外部变量和函数即可。

2) 程序清单

```
1.  #include<stdio.h>
```

```
2. extern int x, y;
3. extern char ch;
4. void func1(void);
```

3) 测试

(a) 测试数据:

表 4-3 编程题 3 的测试数据

测试用例	程序输入	理论结果
用例	A	in file1 x=10,y=20,ch is A in file2 x=11,y=21,ch is B

(b) 对应测试数据的运行结果截图



图 4-10 编程题 3 的测试用例的运行结果

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。

4.3 实验小结

在这次上机实验中，我熟悉了文件包含、宏定义、条件编译和 `assert` 宏的使用；练习了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。熟悉多文件编译技术。以下是我的实验经历和体会

1.宏的定义和使用：在这一部分的实验中，我们尝试调试和修改一个使用宏来计算平方差和交换两个数的源程序。在初始阶段，我遇到了一些常见问题。首先，我注意到宏定义中没有加括号引起的运算错误，其次我发现了宏定义中使用的变量 `t` 在主函数中没有声明，通过仔细检查编译器的错误信息，我能够找到并修复这些问题。

2.通过 `ifdef` 实现选择：通过 `ifdef,endif` 等语句，方便了在一个程序中实现多种功能，使用时只需更改宏的值，增加了程序的可读性和多用途。

3.了解了更多数学函数：在跟踪调试题中，我们需要对结果进行四舍五入，借此机会我了解了 `round`, `floor`, `ceil` 等与取整有关的函数使用方法。

4.熟悉了多文件操作：通过实验题，我了解了多文件函数中头文件如何编写，函数的声明和外部变量声明，对宏有了更多的了解。

实验 5 编译预处理实验

5.1 实验目的

- (1) 掌握数组的说明、初始化和使用。
- (2) 掌握一维数组作为函数参数时实参和形参的用法。
- (3) 掌握字符串处理函数的设计，包括串操作函数及数字串与数之间转换函数实现算法。
- (4) 掌握基于分治策略的二分查找算法和选择法排序算法的思想，以及相关算法的实现。

5.2 实验内容及要求

1、源程序改错与跟踪调试

在下面所给的源程序中，函数 `strcate(t,s)` 的功能是将字符串 `s` 连接到字符串 `t` 的尾部；函数 `strdelc(s,c)` 的功能是从字符串 `s` 中删除所有与给定字符 `c` 相同的字符，程序应该能够输出如下结果：

Programming Language

ProgrammingLanguage Language

ProgrammingLnguge

跟踪和分析源程序中存在的问题，排除程序中的各种逻辑错误，使之能够输出正确的结果。

- (1) 单步执行源程序。进跟踪进入 `strcate` 时，观察字符数组 `t` 和 `s` 中的内容，分析结果是否正确。当单步执行光条刚落在第二个 `while` 语句所在行时，`i` 为何值？`t[i]` 为何值？分析该结果是否存在问题。当单步执行光条落在 `strcate` 函数块结束标记即右花括号 “`}`” 所在行时，字符数组 `t` 和 `s` 分别为何值？分析是否实现了字符串连接。
- (2) 跟踪进入函数 `strdelc` 时，观察字符数组 `s` 中的内容和字符 `c` 的值，分析结果是否正确。单步执行 `for` 语句过程中，观察字符数组 `s`, `j` 和 `k` 值的变化，分析该结果是否存在问题。当单步执行光条落在 `strdelc` 函数块结束标记 “`}`” 所

在行时，字符串 s 为何值？分析是否实现了所要求的删除操作。

/*实验 5 程序改错与跟踪调试题程序*/

```

1.  #include<stdio.h>
2.  void strcate(char [],char []);
3.  void strdelc(char [],char );
4.  int main(void)
5.  {
6.      char a[]="Language", b[]="Programming";
7.      printf("%s %s\n", b,a);
8.      strcate(b,a); printf("%s %s\n",b,a);
9.      strdelc(b, 'a'); printf("%s\n",b);
10. return 0;
11. }
12. void strcate(char t[],char s[])
13. {
14.     int i = 0, j = 0;
15.     while(t[i++]) ;
16.     while((t[i++] = s[j++] )!= '\0');
17. }
18. void strdelc(char s[], char c)
19. {
20.     int j,k;
21.     for(j=k=0; s[j] != '\0'; j++)
22.         if(s[j] != c) s[k++] = s[j];
23. }
    
```

解答：

(1)

1) t,s 中的内容分别为"Programming", "Language"。

名称	值	类型
t	0x00007ff65d10d078 "Programming"	char *
s	0x00007ff65d10d068 "Language"	char *
i	0	int
j	0	int

图 5-1 源程序改错与跟踪调试题的示意图 1

2) 当单步执行光条刚落在第二个 while 语句所在行时，i 为 12，t[i]为 0。

该结果存在问题，因为"Programming"共 11 个字符，t[12]已经越界，此时 i 应该等于 11 才符合预计程序功能。

名称	值	类型
t	0x00007ff7885dd078 "Programming"	char *
s	0x00007ff7885dd068 "Language"	char *
i	12	int
j	0	int
t[i]	0 '\0'	char
s[i])	未定义标识符 "s"	

图 5-2 源程序改错与跟踪调试题的示意图 2

- 3) t,s 中的内容分别为"Programming", "Language", 没有发生改变。可见没有发生字符串的连接, 因为 t[11]为终止符, 在其后的赋值操作是无效的, 均为 0。

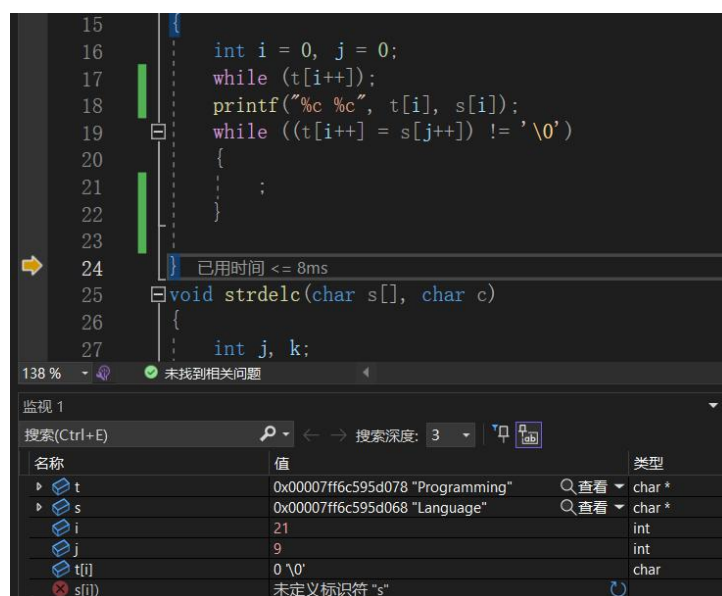


图 5-3 源程序改错与跟踪调试题的示意图 3

(2)

- 1) 观察数组变化, 可见执行了删除操作。

s	0x00007ff64725d078 "Progrmmming"	char *
j	6	int
k	6	int
c	97 'a'	char

图 5-4 源程序改错与跟踪调试题的示意图 4

- 2) 当单步执行光条落在 strdelc 函数块结束标记 “}” 所在行时, 字符串 s 为 "Progrmmingg"。

没有实现期望的删除操作，原因是删除 a 后结尾进行了 s[9]=s[10],而 s[9]应放置终止符。

s	0x00007ff6348fd078 "Progrmmingg"	查看	char *
j	10		int
k	10		int
c	97 'a'		char

图 5-5 源程序改错与跟踪调试题的示意图 5

(3) 修改后的源程序为:

```

1. #include<stdio.h>
2. void strcat(char[], char[]);
3. void strdelc(char[], char);
4. char a[] = "Language", b[] = "Programming";
5. int main(void)
6. {
7.
8.     printf("%s %s\n", b, a);
9.     strcat(b, a); printf("%s %s\n", b, a);
10.    strdelc(b, 'a'); printf("%s\n", b);
11.    return 0;
12. }
13. void strcat(char t[], char s[])//programming Language
14. {
15.     int i = 0, j = 0;
16.     while (t[i++]);
17.     i--;
18.     while ((t[i++] = s[j++]) != '\0')
19.     {
20.         ;
21.     }
22. }
23. void strdelc(char s[], char c)
24. {
25.     int j, k;
26.     for (j = k = 0; s[j] != '\0'; j++)
27.     {
28.         if (s[j] != c) s[k++] = s[j];
29.     }
30.     s[k] = '\0';
31.
32. }

```

修改后的运行结果:



图 5-6 错误修改后的程序运行结果示意图

2、源程序完善和修改替换

(1) 下面的源程序用于求解瑟夫问题：M 个人围成一圈，从第一个人开始依次从 1 至 N 循环报数，每当报数为 N 时报数人出圈，直到圈中只剩下一个人为止。

①请在源程序中的下划线处填写合适的代码来完善该程序。

```

1.  #include<stdio.h>
2.  #define M 10
3.  #define N 3
4.  int main(void)
5.  {
6.  int a[M], b[M]; /* 数组 a 存放圈中人的编号，数组 b 存放出圈人的
    编号 */
7.  int i, j, k;
8.  for(i = 0; i < M; i++) /* 对圈中人按顺序编号 1-M */
9.  a[i] = i + 1;
10. for(i = M, j = 0; i > 1; i--){
11. /* i 表示圈中人个数，初始为 M 个，剩 1 个人时结束循环；j 表示当前
    报数人的位置 */
12. for(k = 1; k <= N; k++) /* 1 至 N 报数 */
13. if(++j > i - 1) j = 0; /* 最后一个人报数后第一个人接着报，形成
    一个圈 */
14. b[M-i] = j ? _____:_____； /* 将报数为 N 的人的编号
    存入数组 b */
15. if(j)
16. for(k = --j; k < i; k++) /* 压缩数组 a，使报数为 N 的人出圈 */
17. _____；
18. }
19. for(i = 0; i < M-1; i++) /* 按次序输出出圈人的编号 */
20. printf("%6d", b[i]);
21. printf("%6d\n", a[0]); /* 输出圈中最后一个人的编号 */
22. return 0;
23. }
    
```

解答:

依次填入 $a[j - 1] : a[i - 1]; a[k] = a[k + 1];$



图 5-7 源程序完善后的程序运行结果示意图

②上面的程序中使用数组元素的值表示圈中人的编号，故每当有人出圈时都要压缩数组，这种算法不够精炼。如果采用做标记的办法，即每当有人出圈时对相应数组元素做标记，从而可省掉压缩数组的时间，这样处理效率会更高一些。请采用做标记的办法修改程序，并使修改后的程序与原程序具有相同的功能。

解答:

修改后的程序如下:

```

1.  #include <stdio.h>
2.  #define M 10
3.  #define N 3
4.
5.  int main() {
6.      int a[M + 1] = { 0 };
7.      int count = 0;
8.      int x = 0;
9.
10.     while (count < M - 1)
11.     {
12.         int step = 0;
13.         while (step < N) {
14.             x = x % M + 1;
15.
16.             if (!a[x]) {
17.                 step++;
18.             }
19.         }
20.         a[x] = 1;
21.         printf("%6d", x);
22.         count++;
23.     }
24.     for (int i = 1; i <= M; i++) {
25.         if (!a[i]) {
26.             printf("%6d", i);
27.             break;

```

```

28.     }
29. }
30.     return 0;
31. }

```

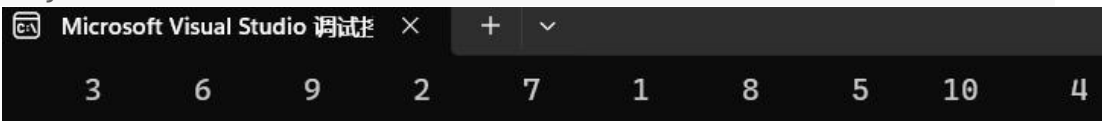


图 5-8 源程序修改后的程序运行结果示意图

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. 程序设计

(1) 输入一个整数，将它在内存中二进制表示的每一位转化成对应的数字字符并且存放到一个字符数组中，然后输出该整数的二进制表示。

解答：

1) 算法流程如图 5.9 所示。

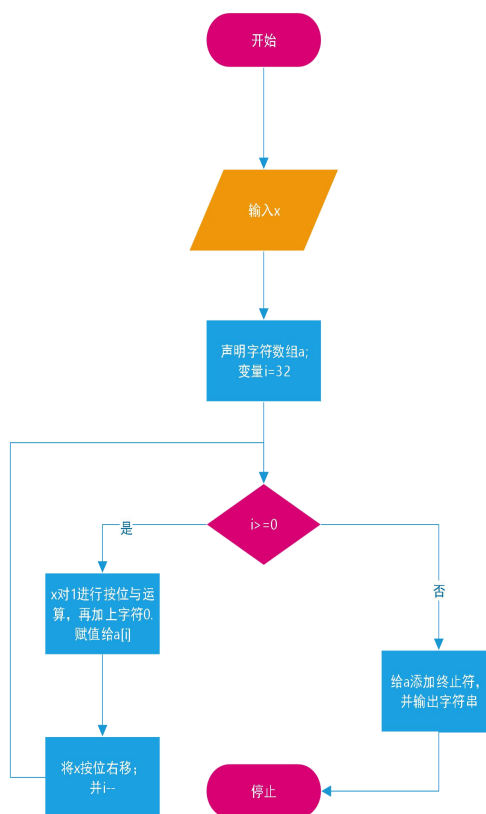


图 5-9 编程题 1 的程序流程图

2) 源程序清单

```

1. #include <stdio.h>

```


(2) 编写一个 C 程序，要求采用模块化程序设计思想，将相关功能用函数实现，并提供菜单选项。该程序具有以下功能：

- ① “成绩输入”，输入 n 个学生的姓名和 C 语言课程的成绩。
- ② “成绩排序”，将成绩按从高到低的次序排序，姓名同时进行相应调整。
- ③ “成绩输出”，输出排序后所有学生的姓名和 C 语言课程的成绩。

④ “成绩查找”，输入一个 C 语言课程成绩值，用二分查找进行搜索。如果查找到有该成绩，则输出该成绩学生的姓名和 C 语言课程的成绩；否则，输出提示 “not found!”。

解答：

1) 解题思路：

1.设置学生结构体，并创建改结构体的数组（指针）

2.功能函数定义与声明：

- ① 输入函数：利用指针从键盘读取数据到对应的位置。
- ② 排序函数：由于数据量较小，利用冒泡排序按成绩排序即可。
- ③ 输出函数：利用 for 循环 printf 输出学生姓名和成绩即可。
- ④ 查找函数：利用二分查找查找对应的值，值得注意的是，当多名学生成绩相同时，我们可以在 mid 两侧继续查找，输出成绩相同的学生。

3. 主函数部分，初始化 choice 为 5，接着输入 choice。

- ① 如果 choice 为 1，调用函数进行数据录入
- ② 如果 choice 为 2，调用函数进行归并排序
- ③ 如果 choice 为 3，调用函数进行姓名、成绩的输出
- ④ 如果 choice 为 4，调用函数进行二分查找
- ⑤ 如果 choice 为 0，则跳出 while 循环

4.输入 0，程序结束。

2) 源程序清单

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>
4.  // 定义学生结构体
5.  struct Student {
```

```

6.     char name[50];
7.     int score;
8. };
9. // 函数原型
10. void inputScores(struct Student** students, int n);
11. void sortScores(struct Student* students, int n);
12. void outputScores(struct Student* students, int n);
13. void searchScore(struct Student* students, int n, int score);
14.
15. int main() {
16.     int choice = 5;
17.     int n;
18.     int searchValue;
19.     struct Student* students = NULL;
20.     while (choice != 0) {
21.         scanf("%d", &choice);
22.         switch (choice) {
23.             case 1:
24.                 scanf("%d", &n);
25.                 students = (struct Student*)malloc(n * sizeof
(struct Student)); // 分配内存空间
26.                 inputScores(&students, n);
27.                 break;
28.             case 2:
29.                 sortScores(students, n);
30.                 break;
31.             case 3:
32.                 outputScores(students, n);
33.                 break;
34.             case 4:
35.                 scanf("%d", &searchValue);
36.                 searchScore(students, n, searchValue);
37.                 break;
38.             case 0:
39.                 return 0;
40.                 break;
41.         }
42.     }
43.     // 释放动态分配的内存
44.     free(students);
45.     return 0;
46. }
47.

```



```

48. // 输入学生姓名和成绩
49. // 指向结构数组的指针
50. void inputScores(struct Student** students, int n) {
51.     for (int i = 0; i < n; i++) {
52.         scanf("%s", (*students)[i].name, 20);
53.         scanf("%d", &(*students)[i].score);
54.     }
55.     printf("%d records were input!\n", n);
56. };
57. // 冒泡排序, 按成绩排序
58. void sortScores(struct Student* students, int n)
59. {
60.     struct Student t;
61.     for (int i = 0; i < n - 1; i++)
62.     {
63.         for (int j = 0; j < n - 1 - i; j++)
64.         {
65.             if (students[j].score < students[j + 1].score
66. )
67.             {
68.                 // 交换成绩和姓名
69.                 t = students[j];
70.                 students[j] = students[j + 1];
71.                 students[j + 1] = t;
72.             }
73.         }
74.         printf("Reorder finished!\n");
75.     }
76. // 输出学生姓名和成绩
77. void outputScores(struct Student* students, int n)
78. {
79.     for (int i = 0; i < n; i++)
80.     {
81.         printf("%s %d\n", students[i].name, students[i].s
82. core);
83.     }
84. // 查找成绩
85. void searchScore(struct Student* students, int n, int sco
86. re) {
87.     int low = 0, high = n - 1, mid;
88.     int found = 0;
89.     while (low <= high) {

```

```

89.         mid = (low + high) / 2;
90.
91.         if (students[mid].score == score) {
92.             printf("%s %d\n", students[mid].name, score);
93.             found = 1;
94.
95.             // 继续在两边查找是否有分数相同的
96.             int i = mid - 1;
97.             while (i >= 0 && students[i].score == score)
98.             {
99.                 printf("%s %d\n", students[i].name, score)
100.            ;
101.                i--;
102.            }
103.            i = mid + 1;
104.            while (i < n && students[i].score == score) {
105.                printf("%s %d\n", students[i].name, score)
106.            ;
107.                i++;
108.            }
109.            break;
110.        }
111.        else if (students[mid].score > score) {
112.            low = mid + 1;
113.        }
114.        else {
115.            high = mid - 1;
116.        }
117.    }
118.    if (!found) {
119.        printf("not found!\n");
120.    }

```

3) 测试

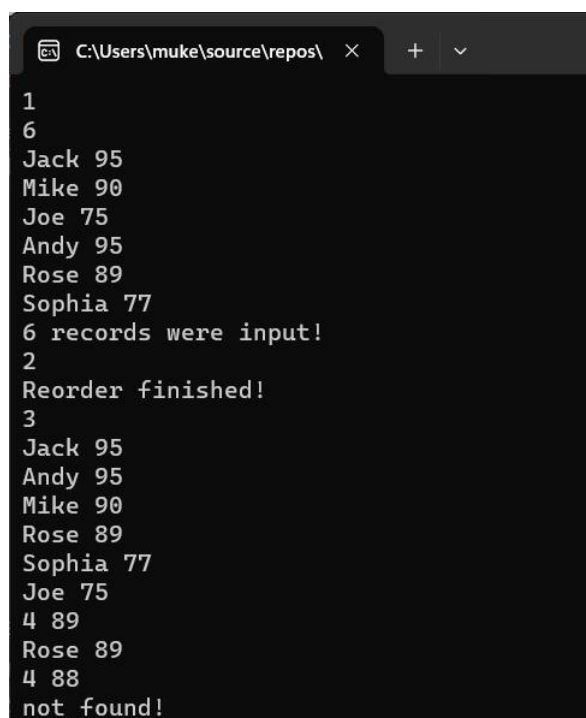
(a) 测试数据:

表 5-2 编程题 2 的测试数据

测试输入	预期输出
1 6 Jack 95 Mike 90 Joe 75 Andy 95 Rose 89 Sophia 77	6 records were input!

2	Reorder finished!
3	Jack 95 Andy 95 Mike 90 Rose 89 Sophia 77 Joe 75
4 89	Rose 89
4 88	not found!

(b) 对应测试数据的运行结果截图



```

C:\Users\muke\source\repos\
1
6
Jack 95
Mike 90
Joe 75
Andy 95
Rose 89
Sophia 77
6 records were input!
2
Reorder finished!
3
Jack 95
Andy 95
Mike 90
Rose 89
Sophia 77
Joe 75
4 89
Rose 89
4 88
not found!
    
```

图 5-12 编程题 2 的测试用例的运行结果

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) 求解 N 皇后问题，即在 $N \times N$ 的棋盘上摆放 N 个皇后，要求任意两个皇后不能在同一行、同一列、同一对角线上。输入棋盘的大小 N (N 取值 1-10)，如果能满足摆放要求，则输出所有可能的摆放法的数量，否则输出“无解。”

解答：

1) 解题思路：

1.定义检查函数：分别检查行，列，对角线上是否有其它皇后。

2.定义解题函数：采用递归回溯思想。For 循环遍历列，递归到下一行，并且递归完后进行回溯开始下一个位置的计算。

3.主函数部分：输入 N，调用解题函数，输出结果。

2) 源程序清单

```
1.  #include <stdio.h>
2.  #include <stdbool.h>
3.  #define MAX_N 10
4.
5.  int N;
6.  int board[MAX_N][MAX_N];
7.
8.  bool isSafe(int row, int col) {
9.
10.     for (int i = 0; i < row; i++) {
11.         if (board[i][col]) {
12.             return false;
13.         }
14.     }
15.
16.     for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
17.     {
18.         if (board[i][j]) {
19.             return false;
20.         }
21.     }
22.
23.     for (int i = row, j = col; i >= 0 && j < N; i--, j++)
24.     {
25.         if (board[i][j]) {
26.             return false;
27.         }
28.     }
29.     return true;
30. }
31.
32. int solveNQueens(int row) {
33.
34.     if (row == N) {
35.         return 1;
36.     }
37.
38.     int count = 0;
39.
40.     for (int col = 0; col < N; col++) {
41.         if (isSafe(row, col)) {
```

```

42.         board[row][col] = 1;
43.         count += solveNQueens(row + 1);
44.         board[row][col] = 0;
45.     }
46. }
47.
48.     return count;
49. }
50.
51. int main() {
52.     scanf_s("%d", &N);
53.
54.     for (int i = 0; i < N; i++) {
55.         for (int j = 0; j < N; j++) {
56.             board[i][j] = 0;
57.         }
58.     }
59.
60.     int solutions = solveNQueens(0);
61.
62.     if (solutions == 0) {
63.         printf("无解\n");
64.     }
65.     else {
66.         printf("%d\n", solutions);
67.     }
68.
69.     return 0;
70. }

```

3) 测试

(a) 测试数据:

表 5-3 编程题 3 的测试数据

测试用例	程序输入	理论结果
用例 1	8	92
用例 2	2	4

(b) 对应测试数据的运行结果截图

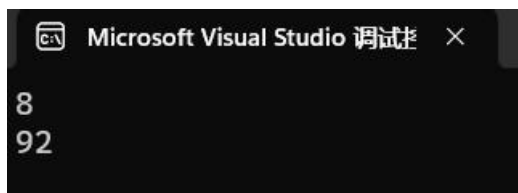


图 5-13 编程题 3 的测试用例 1 的运行结果

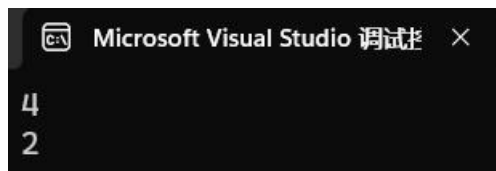


图 5-14 编程题 3 的测试用例 2 的运行结果

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。

5.3 实验小结

在这次编译预处理实验中，我学习了数组的使用、字符串处理函数的设计、基于分治策略的算法实现以及模块化程序设计。以下是对实验过程的体会和心得。

1. **数组的使用。**通过实验，我发现并解决了数组使用过程中的各项问题，如数组的初始化，越界，终止符位置，如何实现动态内存分配。通过调试和修改，最终顺利完成了实验。
2. **学习了经典问题。**约瑟夫环和 N 皇后是典型的编程题目，在约瑟夫环问题中，如何处理 x 的值是个难点。经过摸索，我想到用 $x \% M + 1$ 来实现环状报数。N 皇后问题有多种解法，通过递归回溯的思想，我初步接触了深度优先搜索算法，对 C 语言的应用有了更深刻的理解，并体会到了解决问题的方法和技巧。
3. **模块化设计：**在编写 C 程序时，需要合理设计函数，确保模块化和可维护性。通过尝试编写学生成绩管理系统，我对此有了一定的了解，尝试编写模块化，可读性高，可修改性强的算法。
4. **多方面考虑问题：**在程序设计 2 的成绩搜索中，容易忽略多名学生成绩相同的情况，导致只输出其中一人的姓名和成绩。通过多方面思考，我发现并成功解决了问题。

实验 6 指针实验

6.1 实验目的

- (1) 熟练掌握指针的说明、赋值、使用。
- (2) 掌握用指针引用数组的元素，熟悉指向数组的指针的使用。
- (3) 熟练掌握字符数组与字符串的使用，掌握指针数组及字符指针数组的用法。
- (4) 掌握指针函数与函数指针的用法。
- (5) 掌握带有参数的 main 函数的用法。

6.2 实验内容及要求

1、源程序改错与跟踪调试

在下面所给的源程序中，函数 `strcpy(t, s)` 的功能是将字符串 `s` 复制给字符串 `t`，并且返回串 `t` 的首地址。请单步跟踪程序，根据程序运行时出现的现象或观察到的字符串的值，分析并排除源程序的逻辑错误，使之能按照要求输出如下结果：

Input a string:

programming✓ （键盘输入）

programming

Input a string again:

language✓ （键盘输入）

language

```
1.  #include<stdio.h>
2.  char *strcpy(char *, const char *);
3.  int main(void)
4.  {
5.      char *s1, *s2, *s3;
6.      printf("Input a string:\n", s2);
7.      scanf("%s", s2);
8.      strcpy(s1, s2);
9.      printf("%s\n", s1);
10.     printf("Input a string again:\n", s2);
```

```

11.     scanf("%s", s2);
12.     s3 = strcpy(s1, s2);
13.     printf("%s\n", s3);
14.     return 0;
15. }
16.
17.  /*将字符串 s 复制给字符串 t, 并且返回串 t 的首地址*/
18.  char * strcpy(char *t, const char *s)
19.  {
20.      while(*t++ = *s++);
21.      return (t);
22.  }
    
```

解答:

(1) 错误修改:

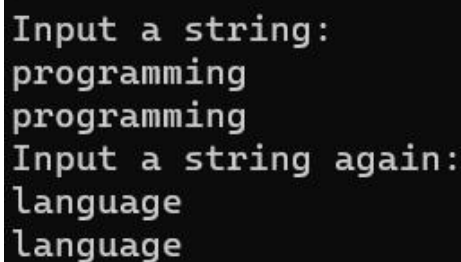
- 1) 第 5 行未分配空间, 应改为 `char s1[100], s2[100], s3[100];`
- 2) 第 12 行 `s3` 不是可修改的左值, 正确形式为: `strcpy(s3, s2);`
- 3) 第 21 行返回的不是字符串 `t` 的首地址, 而是 `t` 进行若干次++后地址,

两者差为(串长+1),正确形式为:

```

char* p = t;
while (*t++ = *s++);
return p;
    
```

(2) 错误修改后运行结果:



```

Input a string:
programming
programming
Input a string again:
language
language
    
```

图 6-1 错误修改后的程序运行结果示意图

2、源程序完善、修改替换题

(1) 下面程序中函数 `strsort` 用于对字符串进行升序排序, 在主函数中输入 `N` 个字符串 (字符串长度不超过 49) 存入通过 `malloc` 动态分配的存储空间, 然

后调用 `strsort` 对这 N 个串按字典序升序排序。

①请在源程序中的下划线处填写合适的代码来完善该程序。

```

1.  #include<stdio.h>
2.  #include<_____>
3.  #include<string.h>
4.  #define N 4
5.  /*对指针数组 s 指向的 size 个字符串进行升序排序*/
6.  void strsort(char *s[], int size)
7.  {
8.      _____temp;
9.      int i, j;
10.     for(i=0; i<size-1; i++)
11.         for (j=0; j<size-i-1; j++)
12.             if (_____)
13.             {
14.                 temp = s[j];
15.                 _____;
16.                 s[j+1] = temp;
17.             }
18.     }
19.
20.     int main()
21.     {
22.         int i;
23.         char *s[N], t[50];
24.         for (i=0; i<N; i++)
25.         {
26.             gets(t);
27.             s[i] = (char *)malloc(strlen(t)+1);
28.             strcpy(_____);
29.         }
30.         strsort(_____);
31.         for (i=0; i<N; i++) {puts(s[i]); free(s[i]);}
32.         return 0;
33.     }

```

解答：

(1) 填空后的程序为

```

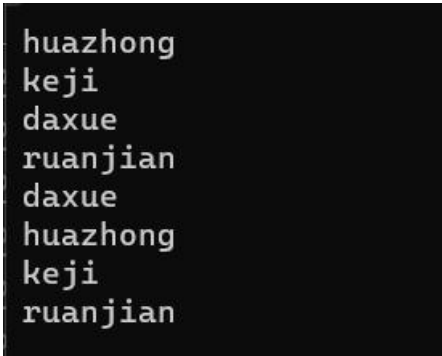
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  #include<string.h>
4.  #define N 4
5.  /*对指针数组 s 指向的 size 个字符串进行升序排序*/

```

```

6.  void strsort(char* s[], int size)
7.  {
8.      char *temp;
9.      int i, j;
10.     for (i = 0; i < size - 1; i++)
11.         for (j = 0; j < size - i - 1; j++)
12.             if (strcmp(s[j],s[j+1]))
13.                 {
14.                     temp = s[j];
15.                     s[j]=s[j+1];
16.                     s[j + 1] = temp;
17.                 }
18.     }
19.
20. int main()
21. {
22.     int i;
23.     char* s[N], t[50];
24.     for (i = 0; i < N; i++)
25.     {
26.         gets_s(t,50);
27.         s[i] = (char*)malloc(strlen(t) + 1);
28.         strcpy(s[i],t);
29.     }
30.     strsort(s,N);
31.     for (i = 0; i < N; i++) { puts(s[i]); free(s[i]); }
32.     return 0;
33. }
    
```

(2) 程序完善后运行结果:



```

huazhong
keji
daxue
ruanjian
daxue
huazhong
keji
ruanjian
    
```

图 6-2 源程序完善后的程序运行结果示意图

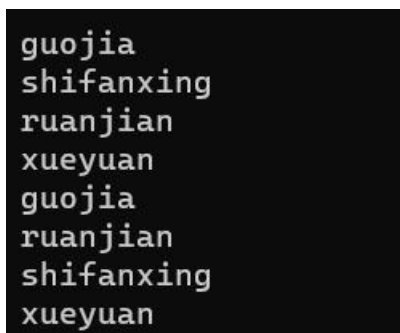
②数组作为函数参数其本质类型是指针。例如，对于形参 `char *s[]`，编译器将其解释为 `char **s`，两种写法完全等价。请用二级指针形参重写 `strsort` 函数，并且在该函数体的任何位置都不允许使用下标引用。

解答：

(1) 修改后的函数为

```
1. void strsort(char**s, int size)
2. {
3.     char* temp;
4.     int i, j;
5.
6.     for (i = 0; i < size - 1; i++)
7.         for (char **p=s, j=0; j < size - i - 1; j++, p++)
8.             if (strcmp(*p, *(p+1))>0)
9.             {
10.                temp = *p;
11.                *p = *(p+1);
12.                *(p+1)= temp;
13.            }
14. }
```

(2) 程序修改后运行结果：



```
guojia
shifanxing
ruanjian
xueyuan
guojia
ruanjian
shifanxing
xueyuan
```

图 6-3 源程序修改后的程序运行结果示意图

(2) 下面源程序通过函数指针和菜单选择来调用库函数实现字符串操作：串复制 `strcpy`、串连接 `strcat` 或串分解 `strtok`。

①请在源程序中的下划线处填写合适的代码来完善该程序，使之能按照要求输出下面结果：

1 copy string.

2 connect string.

3 parse string.

4 exit.

input a number (1-4) please!

2✓ （键盘输入）

input the first string please!

the more you learn,✓ （键盘输入）

input the second string please!

the more you get. ✓ （键盘输入）

the result is the more you learn, the more you get.

```

1.  # include<stdio.h>
2.  # include<string.h>
3.  int main (void)
4.  {
5.      _____;
6.  char a[80], b[80], *result;
7.  int choice;
8.  while(1)
9.  {
10.     do
11.     {
12.         printf("\t\t1 copy string.\n");
13.         printf("\t\t2 connect string.\n");
14.         printf("\t\t3 parse string.\n");
15.         printf("\t\t4 exit.\n");
16.         printf("\t\tinput a number (1-4) please.\n");
17.         scanf("%d", &choice);
18.     }while(choice<1 || choice>4);
19.     switch(choice)
20.     {
21.         case 1: p = strcpy; break;
22.         case 2: p = strcat; break;
23.         case 3: p = strtok; break;
24.         case 4: goto down;
25.     }
26.     getchar();
27.     printf("input the first string please!\n");
28.     _____;
29.     printf("input the second string please!\n");

```

```

30. _____;
31. result = _____(a, b);
32. printf("the result is %s\n", result);
33. }
34. down:
35. return 0;
36. }

```

解答:

(1) 完善后的函数为

```

1. #include<stdio.h>
2. #include<string.h>
3.
4. int main(void)
5. {
6.     char* (*p)(char a[], const char b[])=NULL;
7.     char a[80], b[80], * result;
8.     int choice;
9.     while (1)
10.    {
11.        do
12.        {
13.            printf("\t\t1 copy string.\n");
14.            printf("\t\t2 connect string.\n");
15.            printf("\t\t3 parse string.\n");
16.            printf("\t\t4 exit.\n");
17.            printf("\t\tinput a number (1-4) please.\n");
18.            scanf("%d", &choice);
19.        } while (choice < 1 || choice>4);
20.        switch (choice)
21.        {
22.            case 1: p = strcpy; break;
23.            case 2: p = strcat; break;
24.            case 3: p = strtok; break;
25.            case 4: goto down;
26.        }
27.        getchar();
28.        printf("input the first string please!\n");
29.        gets_s(a, 80);
30.        printf("input the second string please!\n");
31.        gets_s(b, 80);
32.        result = p(a, b);
33.        printf("the result is %s\n", result);
34.    }

```

```

35. down:
36.     return 0;
37. }

```

(2) 程序完善后运行结果:

```

1 copy string.
2 connect string.
3 parse string.
4 exit.
input a number (1-4) please.
2
input the first string please!
the more you learn,
input the second string please!
the more you get.
the result is the more you learn,the more you get.

```

图 6-4 源程序完善后的程序运行结果示意图

②函数指针的一个用途是用户散转程序,即通过一个转移表(函数指针数组)来实现多分枝函数处理,从而省去了大量的 if 语句或者 switch 语句。转移表中存放了各个函数的入口地址(函数名),根据条件的设定来查表选择执行相应的函数。请使用转移表而不是 switch 语句重写以上程序。

解答:

(1) 修改后的函数为

```

1.  #include<stdio.h>
2.  #include<string.h>
3.  #include<stdlib.h>
4.
5.  int main(void)
6.  {
7.      char* (*p[3])(char a[], const char b[]) = { strcpy,st
rcat,strtok,};
8.      char a[80], b[80], * result;
9.      int choice;
10.     while (1)
11.     {         do

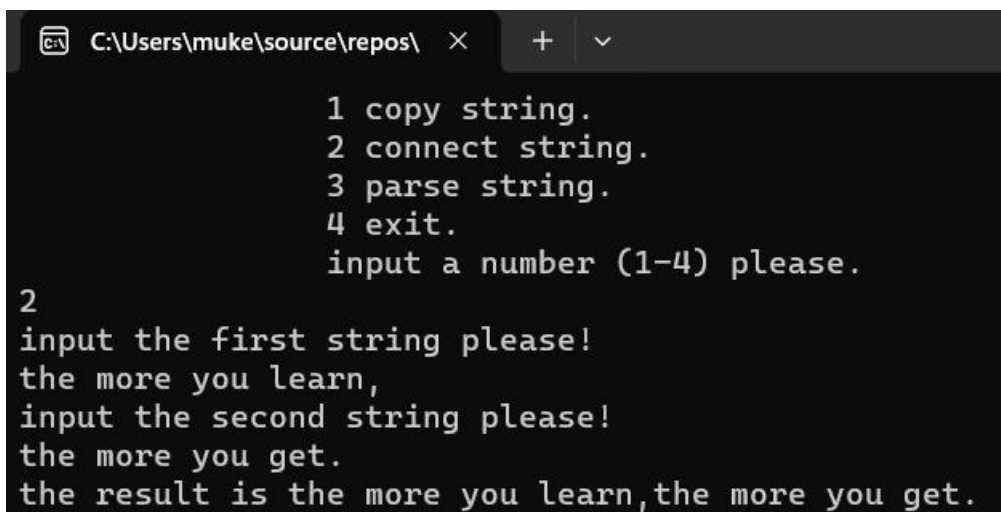
```

```

12.      {
13.          printf("\t\t1 copy string.\n");
14.          printf("\t\t2 connect string.\n");
15.          printf("\t\t3 parse string.\n");
16.          printf("\t\t4 exit.\n");
17.          printf("\t\tinput a number (1-4) please.\n");
18.          scanf("%d", &choice);
19.      } while (choice < 1 || choice>4);
20.      if (choice == 4) return 0;
21.      else {
22.          getchar();
23.          printf("input the first string please!\n");
24.          gets_s(a, 80);
25.          printf("input the second string please!\n");
26.          gets_s(b, 80);
27.          result = p[choice -1](a, b);
28.          printf("the result is %s\n", result);
29.      }
30.  }
31.
32.      return 0;
33.  }

```

(2) 程序修改后运行结果:



```

C:\Users\muke\source\repos\
1 copy string.
2 connect string.
3 parse string.
4 exit.
input a number (1-4) please.
2
input the first string please!
the more you learn,
input the second string please!
the more you get.
the result is the more you learn,the more you get.

```

图 6-5 源程序修改后的程序运行结果示意图

3.程序设计

(1) 指定 main 函数的参数

在 IDE（比如 DevC++）中，选择“运行” | “参数”菜单，在“传递给主程序的参数”文本框中输入 main 函数的参数 arg1 arg2 arg3，只输入命令行中文件名后的参数，文件名不作为参数输入，参数间以空格隔开。编写程序在命令行输出这三个参数。（注意不同 IDE 输入参数的方式不相同，可参考各个 IDE 的使用手册。）

解答：

1) 解题思路：

Visual Studio 点击调试，工程属性，配置属性，调试，命令行参数，例如输入 1 2 3，确定，应用到工程中

2) 源程序清单

```

1.  # include<stdio.h>
2.  int main(int argc, char* argv[]) {
3.      // argc 表示命令行参数的数量
4.      // argv 是一个指向参数字符串数组的指针，其中 argv[0] 存储
      程序的名称，后续元素存储传递给程序的参数
5.      // 注意：IDE 中配置的参数通常不包括程序名称，因此
      从 argv[1] 开始获取实际的参数
6.
7.      // 输出参数数量
8.      printf("Number of command-line arguments: %d\n", argc
      - 1);
9.
10.     // 输出每个参数
11.     for (int i = 1; i < argc; i++) {
12.         printf("Argument %d: %s\n", i, argv[i]);
13.     }
14.     return 0;
15. }
```

3) 测试

(a) 测试数据：

表 6-1 编程题 1 的测试数据

测试用例	命令行输入	理论结果
用例	1 2 3	3 1 2 3

```
Number of command-line arguments: 3
Argument 1: 1
Argument 2: 2
Argument 3: 3
```

图 6-6 编程题 1 的测试用例的运行结果

(2) 一个长整型变量占 4 个字节，其中每个字节又分成高 4 位和低 4 位。输入一个长整型变量，要求从高字节开始，依次取出每个字节的高 4 位和低 4 位并以十六进制数字字符的形式进行显示，通过指针取出每字节。

样例输入：15

样例输出：0000000F

解答：

1) 解题思路：

1. 声明一个指向无符号字符的指针 ptr，用于访问每个字节。
2. 将 num 的地址强制转换为 unsigned char* 类型，以允许以字节为单位访问变量。
3. 计算长整型变量的字节大小，并减去 1，以获得指向最高字节的指针。
4. 循环遍历 long int 的每个字节
5. 利用 00001111==位掩码 0X0F 取出高四位和低四位
6. 递减指针 ptr，以访问下一个字节。
7. 结束

2) 源程序清单

```
1. #include<stdio.h>
2.
3. int main() {
4.     int num;
5.     scanf("%d", &num);
6.     //声明一个指向无符号字符的指针 ptr，用于访问每个字节。
```

```

7.      //将 num 的地址强制转换为 unsigned char* 类型, 以允许以字
      节为单位访问变量。
8.      unsigned char* p = (unsigned char*)&num + sizeof(int)
      - 1;
9.      //计算长整型变量的字节大小, 并减去 1, 以获得指向最高字节的指
      针。
10.     //循环遍历 long int 的每个字节
11.     for (int i = 0; i < sizeof(int); i++) {
12.         //00001111==位掩码 0X0F
13.         printf("%X", (*p >> 4) & 0x0F);
14.         printf("%X", *p & 0x0F); //递减指针 ptr, 以访问下一个
      字节。
15.         p--;
16.     }
17.     printf("\n");
18.
19.     return 0;
20. }

```

3) 测试

(a) 测试数据:

表 6-2 编程题 2 的测试数据

测试用例	程序输入	理论结果
用例 1	15	0000000F
用例 2	2147483647	7FFFFFFF

(b) 对应测试数据的运行结果截图

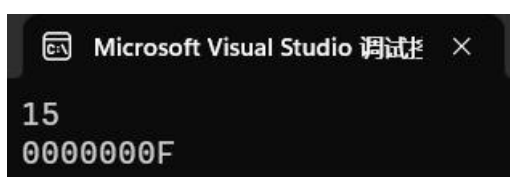


图 6-7 编程题 2 的测试用例 1 的运行结果



图 6-8 编程题 2 的测试用例 2 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) 旋转是图像处理的基本操作, 编程实现一个将一个图像逆时针旋转 90° 。

提示：计算机中的图像可以用一个矩阵来表示，旋转一个图像就是旋转对应的矩阵。将旋转矩阵的功能定义成函数，通过使用指向数组元素的指针作为参数使该函数能处理任意大小的矩阵。要求在 `main` 函数中输入图像矩阵的行数 `n` 和列数 `m`，接下来的 `n` 行每行输入 `m` 个整数，表示输入的图像。输出原始矩阵逆时针旋转 90° 后的矩阵。

样例输入：

```
2 3
1 5 3
3 2 4
```

样例输出：

```
3 4
5 2
1 3
```

解答：

2) 解题思路：

1. 创建两个足够大的数组，分别储存输入和输出矩阵
2. 输入 `m`, `n` 和初始矩阵
3. 定义旋转函数，通过寻找变换后坐标与变换前坐标的关系，对输出矩阵的各元素赋值

4. 调用旋转函数
5. 打印旋转后的矩阵
6. 结束

3) 源程序清单

```
1. #include<stdio.h>
2. void rotate(int input[20][20], int output[20][20], int n,
   int m) {
3.     for (int i = 0; i < m; i++) {
4.         for (int j = 0; j < n; j++) {
5.             output[n - 1 - j][i] = input[i][j];
6.         }
7.     }
8. }
9. int main() {
```

```
10.     int n, m;
11.     scanf("%d %d", &m, &n);
12.     int input[20][20] = { 0 };
13.     int output[20][20] = { 0 };
14.     // 输入图像矩阵
15.     for (int i = 0; i < m; i++) {
16.         for (int j = 0; j < n; j++) {
17.             scanf("%d", &input[i][j]);
18.         }
19.     }
20.     rotate(input, output, n, m);
21.     for (int i = 0; i < n; i++) {
22.         for (int j = 0; j < m; j++) {
23.             printf("%d", output[i][j]);
24.             if (j < m - 1) {
25.                 printf(" ");
26.             }
27.         }
28.         printf("\n");
29.     }
30.     return 0;
31. }
```

3) 测试

(a) 测试数据:

表 6-3 编程题 3 的测试数据

测试用例	程序输入	理论结果
用例 1	2 3	3 4
	1 5 3	5 2
	3 2 4	1 3
用例 2	3 4	4 8 12
	1 2 3 4	3 7 11
	5 6 7 8	2 6 10
	9 10 11 12	1 5 9

(b) 对应测试数据的运行结果截图



```

2 3
1 5 3
3 2 4
3 4
5 2
1 3
    
```

图 6-9 编程题 3 的测试用例 1 的运行结果



```

3 4
1 2 3 4
5 6 7 8
9 10 11 12
4 8 12
3 7 11
2 6 10
1 5 9
    
```

图 6-10 编程题 3 的测试用例 2 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) 输入 n 行文本，每行不超过 80 个字符，用字符指针数组指向键盘输入的 n 行文本，且 n 行文本的存储无冗余，删除每一行中的前置空格（' '）和水平制表符（'\t'）。要求：将删除一行文本中前置空格和水平制表符的功能定义成函数，在 `main` 函数中输出删除前置空格符的各行。

解答：

1) 解题思路：

1. 定义判断制表符函数
2. 声明字符数组，换行判断变量 `LineStart`
3. 循环输入字符并判断，若是制表符，直接获取下一次输入
4. 若不是制表符，判断是否为行首空格，是则 `continue`，否则将该字符存入数组
5. 结束输入后打印字符数组中的值
6. 结束

2) 源程序清单

```

1.  #include <stdio.h>
2.  #include <string.h>
3.  int panduan(char c) {
4.      if (c == '\t') {
5.          return 0;
6.      }
7.      else {
8.          return 1;
9.      }
10. }
11.
12. int main() {
13.     char s[80];
14.     char c;
15.     int i = 0;
16.     int LineStart = 1;
17.
18.     while ((c = getchar()) != EOF) {
19.         if (panduan(c)) {
20.             if (LineStart && c == ' ') {
21.                 continue;
22.             }
23.             s[i++] = c;
24.             if (c == '\n') {
25.                 LineStart = 1;
26.             }
27.             else {
28.                 LineStart = 0;
29.             }
30.         }
31.     }
32.
33.     for (int j = 0; j < i; j++) {
34.         printf("%c", s[j]);
35.     }
36.
37.     return 0;
38. }

```

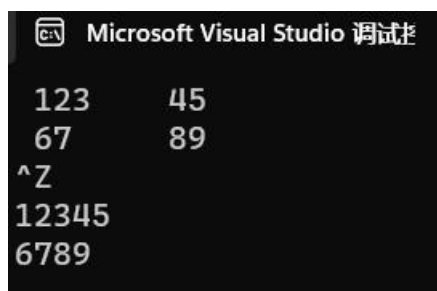
3) 测试

(a) 测试数据:

表 6-4 编程题 4 的测试数据

测试用例	程序输入	理论结果
用例 1	123 45 67 89	12345 6789
用例 2	aab c bcd e	aab c bcde

(b) 对应测试数据的运行结果截图

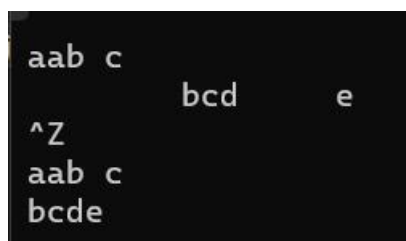


```

Microsoft Visual Studio 调试
123 45
67 89
^Z
12345
6789

```

图 6-11 编程题 4 的测试用例 1 的运行结果



```

aab c
bcd e
^Z
aab c
bcde

```

图 6-12 编程题 4 的测试用例 2 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) 编写 8 个任务函数，一个 scheduler 调度函数和一个 execute 执行函数。仅在 main 函数中调用 scheduler 函数，scheduler 函数要求用最快的方式调度执行用户指定的任务函数。

①先设计 task0, task1, task2, task3, task4, task5, task6, task7 共 8 个任务函数，每个任务函数的任务就是输出该任务被调用的字符串。例如，第 0 个任务函数输出“task0 is called!”，第 1 个任务函数输出“task1 is called!”，以此类推。

②scheduler 函数根据键盘输入的数字字符的先后顺序，一次调度选择对应的任务函数。例如，输入：1350 并回车，则 scheduler 函数一次调度选择 task1, task3, task5, task0，然后以函数指针数组和任务个数为参数将调度选择结果传递给 execute 函数并调用 execute 函数。

③execute 函数根据 scheduler 函数传递的指针数组和任务个数为参数，按照

指定的先后顺序依此调用执行选定的任务函数。

例如，当输入 13607122 并回车，程序运行结果如下：

task1 is called!

task3 is called!

task6 is called!

task0 is called!

task7 is called!

task1 is called!

task2 is called!

task2 is called!

解答：

1) 解题思路：

1. 定义八个任务函数

2. 定义执行函数，传入函数指针数组和整型数组，通过循环依次调用相应的任务函数

3. 定义调度函数，传入输入的字符数组，声明函数指针数组，将字符数组转化为整型数组，并计算出长度，最后调用执行函数

4. 主函数部分:声明字符数组，输入值存入字符数组，调用调度函数。

5. 结束

2) 源程序清单

```
1.  #include <stdio.h>
2.  void task0() { printf("task0 is called!\n"); }
3.  void task1() { printf("task1 is called!\n"); }
4.  void task2() { printf("task2 is called!\n"); }
5.  void task3() { printf("task3 is called!\n"); }
6.  void task4() { printf("task4 is called!\n"); }
7.  void task5() { printf("task5 is called!\n"); }
8.  void task6() { printf("task6 is called!\n"); }
9.  void task7() { printf("task7 is called!\n"); }
10. void execute(void (*tasks[])(), char* Task, int num) {
11.     for (int i = 0; i < num; i++) {
12.         tasks[Task[i]]();
13.     }
14. }
```



```

15. void scheduler(char* input)
16. {
17.     int num = 0;
18.     void (*function[8])() = { task0, task1, task2, task3,
        task4, task5, task6, task7 };
19.     char Task[20] = { 0 };
20.
21.     for (int i = 0; input[i] != '\0'; i++)
22.     {
23.         int task_number = input[i] - '0';
24.         Task[i] = task_number;
25.         if (task_number >= 0 && task_number < 8)
26.         {
27.             num++;
28.         }
29.
30.     }
31.     execute(function, Task, num);
32. }
33.
34. int main() {
35.     char input[20];
36.     scanf("%s", input, 20);
37.     scheduler(input);
38.     return 0;
39. }

```

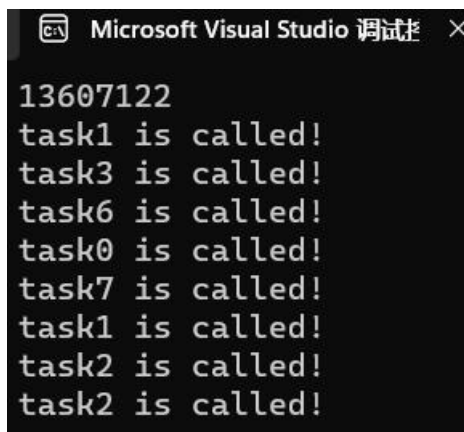
3) 测试

(a) 测试数据:

表 6-5 编程题 5 的测试数据

测试用例	用例 1	用例 2
程序输入	13607122	6735643
理论输出	task1 is called! task3 is called! task6 is called! task0 is called! task7 is called! task1 is called! task2 is called! task2 is called!	task6 is called! task7 is called! task3 is called! task5 is called! task6 is called! task4 is called! task3 is called!

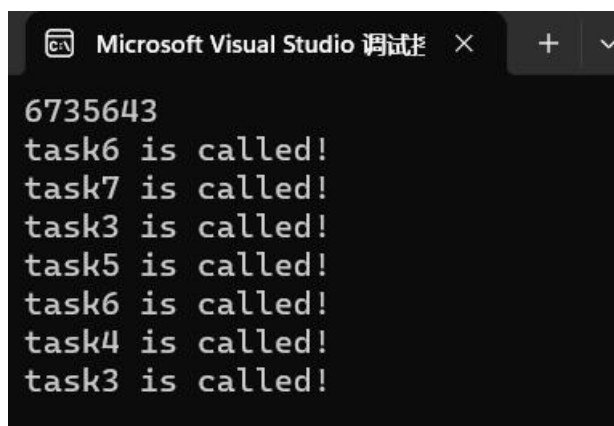
(b) 对应测试数据的运行结果截图



```

Microsoft Visual Studio 调试
13607122
task1 is called!
task3 is called!
task6 is called!
task0 is called!
task7 is called!
task1 is called!
task2 is called!
task2 is called!
    
```

图 6-13 编程题 5 的测试用例 1 的运行结果



```

Microsoft Visual Studio 调试
6735643
task6 is called!
task7 is called!
task3 is called!
task5 is called!
task6 is called!
task4 is called!
task3 is called!
    
```

图 6-14 编程题 5 的测试用例 2 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。

6.3 实验小结

在这次指针实验中，我学习了指针的说明、赋值、使用。掌握用指针引用数组的元素，熟悉指向数组的指针的使用。熟练掌握字符数组与字符串的使用，掌握指针数组及字符指针数组的用法。以下是我发现的一些问题和心得体会。

1. **输入函数的选择。**要输入一个字符串，scanf, gets_s, fget_s 可能会产生不同的效果。例如，scanf 在遇到空格时便会停止，gets_s 遇到回车才停止且不读取回车，而 fgets_s 会读取回车。
2. **函数指针数组的使用。**通过函数指针，我们可以简化代码，提高代码的简洁性，通过这一点的学习，我增强了对指针的理解，体会到了指针的妙处。
3. **注意缓冲区和越界问题。**在使用指针和数组时，对数组的大小要尤其上心，以免发生越界问题，在声明和使用指针时，要注意其类型，以达到理想的效

果，在输入字符串时，要考虑用 `getchar()` 清除缓冲区，以免残留的字符被意外读取。

实验 7 结构与联合实验

7.1 实验目的

- (1) 通过实验，熟悉和掌握结构的说明和引用、结构的指针、结构数组、以及函数中使用结构的方法。
- (2) 通过实验，掌握动态储存分配函数的用法，掌握自引用结构，单向链表的创建、遍历、结点的增删、查找等操作。
- (3) 了解字段结构和联合的用法。

7.2 实验题目及要求

1. 表达式求值的程序验证题

设有说明：

```
char u[]="UVWXYZ";
char v[]="xyz";
struct T{
    int x;
    char c;
    char *t;
}a[]={11, 'A', u}, {100, 'B', v}}, *p=a;
```

请先自己计算下面表达式的值，然后通过编程计算来加以验证。(各表达式相互无关)

序号	表达式	计算值	验证值
1	(++p)->x	100	100
2	p++,p->c	B	B
3	*p++->t,*p->t	x	x
4	*(++p)->t	x	x
5	*++p->t	V	V
6	++*p->t	V	V

解答:

(1) 计算过程:

- 1) ++p 先进行 p+1 再运算, 此时 p 指向数组第二个元素, 原表达式即数组第二个结构中 x 的值 100
- 2) 逗号表达式从左往右计算, 结果取右值, 为 B
- 3) *p++->t: 这是一个后缀递增运算符, 首先取 p->t 的值, 然后递增 p。接着, 通过 * 取得该值的内容 U。*p->t: 这个表达式首先会取 p->t 的值, 然后通过 * 取得该值的内容。所以最终, p 指向第二个结构体的元素 v(指针), 再使用间接引用运算符 * 取值为 x, 表达式为右值 x。
- 4) *(++p)->t: 先进行 p+1 再运算, 再使用间接引用运算符 * 对 p 指向的第二个结构体中的元素 v(指针)进行取值为 x
- 5) *++p->t: 首先, 会访问结构体或联合体的成员 t: 接着, 会对 p->t 的结果进行递增操作, 使指针 t 指向数组 u 第二个元素。注意, 这是一个前缀递增运算符, 所以会在取值之前递增 p->t 最后, 通过前两步得到的结果, 使用间接引用运算符 * 取得其内容。
- 6) ->的优先级大于++和*, 而++和*是右结合, 综合起来, ++*p->t 的操作顺序是先取得 p->t 的值, 然后使用间接引用运算符 * 取得其内容。最后对该内容进行++, 此时数组 u 中第一个元素的值变成 V, 表达式的值即该值: V

(2) 运行程序:

```

1.  int main() {
2.  char u[] = "UVWXYZ";
3.  char v[] = "xyz";
4.  struct T {
5.  int x;
6.  char c;
7.  char* t;
8.  }a[] = { {11, 'A', u}, {100, 'B', v}}, * p = a;
9.
10. printf("%d\n", (++p)->x); p--;
11. printf("%c\n", (p++, p->c)); p--;
12. printf("%c\n", (*p++->t, *p->t)); p--;
13. printf("%c\n", *(++p)->t); p--;
14. printf("%c\n", *++p->t), --p->t;
15. printf("%c\n", ++ * p->t);
16. }
```

(3) 运行结果示意图

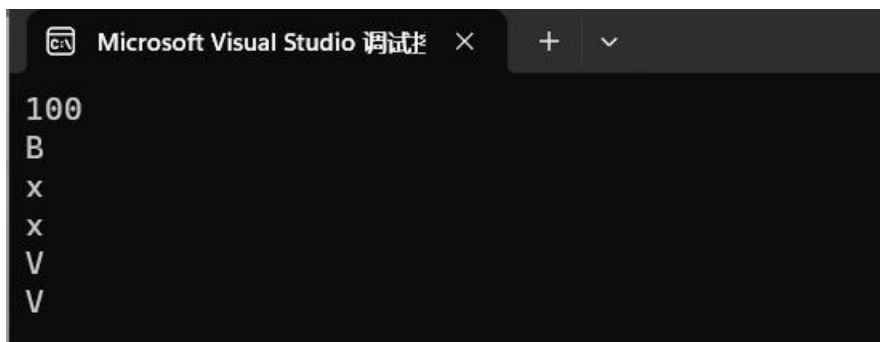


图 7-1 表达式求值的程序运行结果示意图

2. 源程序修改替换题

给定一批整数，以 0 作为结束标志且不作为结点，将其建成一个先进先出的链表，先进先出链表的指头指针始终指向最先创建的结点（链头），先建结点指向后建结点，后建结点始终是尾结点。

（1）源程序中存在什么样的错误（先观察执行结果）？对程序进行修改、调试，使之能够正确完成指定任务。

源程序如下：

```

1.  #include "stdio.h"
2.  #include "stdlib.h"
3.  struct s_list{
4.  int data; /* 数据域 */
5.  struct s_list *next; /* 指针域 */
6.  } ;
7.  void create_list (struct s_list *headp,int *p);
8.  void main(void)
9.  {
10.   struct s_list *head=NULL,*p;
11.   int s[]={1,2,3,4,5,6,7,8,0}; /* 0 为结束标记 */
12.   create_list(head,s); /* 创建新链表 */
13.   p=head; /*遍历指针 p 指向链头 */
14.   while(p){
15.       printf("%d\t",p->data); /* 输出数据域的值 */
16.       p=p->next; /*遍历指针 p 指向下一结点 */
17.   }
18.   printf("\n");

```

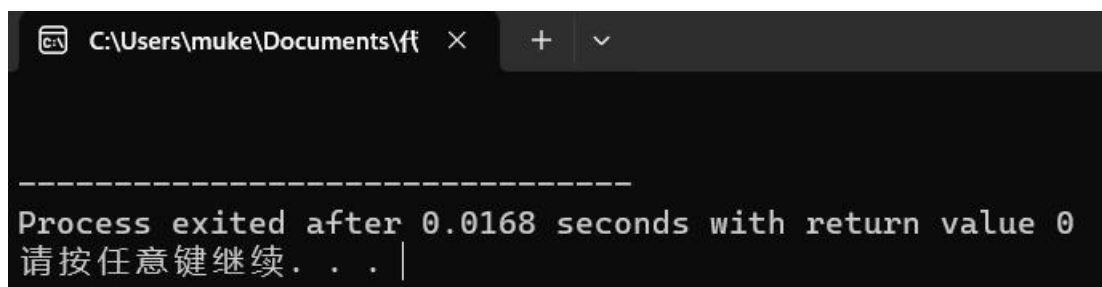
```

19. }
20. void create_list(struct s_list *headp,int *p)
21. {
22.     struct s_list * loc_head=NULL,*tail;
23.     if(p[0]==0) /* 相当于*p==0 */
24.         ;
25.     else { /* loc_head 指向动态分配的第一个结点 */
26.         loc_head=(struct s_list *)malloc(sizeof(struct s_list))
27.         ;
28.         loc_head->data=*p++; /* 对数据域赋值 */
29.         tail=loc_head; /* tail 指向第一个结点 */
30.         while(*p){ /* tail 所指结点的指针域指向动态创建的结点 */
31.             tail->next=(struct s_list *)malloc(sizeof(struct s_list));
32.             tail=tail->next; /* tail 指向新创建的结点 */
33.             tail->data=*p++; /* 向新创建的结点的数据域赋值 */
34.         }
35.         tail->next=NULL; /* 对指针域赋 NULL 值 */
36.         headp=loc_head; /* 使头指针 headp 指向新创建的链表 */
37.     }

```

解答：

1) 错误修改前的运行结果：



```

-----
Process exited after 0.0168 seconds with return value 0
请按任意键继续. . . |

```

图 7-2 源程序修改替换题错误修改前的程序运行结果示意图

程序只输出一个回车，与预期不符。

2) 错误修改：

函数传入指针 head 并对 head 进行赋值，但由于函数的值传递特性，实参的值并没有改变，主函数中的指针 head，p 仍为空指针，因此，函数应传入指针 head 的地址，修改后的程序如下：

```

1.  struct s_list {
2.      int data; /* 数据域 */
3.      struct s_list* next; /* 指针域 */
4.  };
5.  void create_list(struct s_list** headp, int* p);
6.  int main()
7.  {
8.      struct s_list* head = NULL, * p;
9.      int s[] = { 1,2,3,4,5,6,7,8,0 }; /* 0 为结束标记 */
10.     create_list(&head, s); /* 创建新链表 */
11.     p = head; /*遍历指针 p 指向链头 */
12.
13.     while (p) {
14.         printf("%d\t", p->data); /* 输出数据域的值 */
15.         p = p->next; /*遍历指针 p 指向下一结点 */
16.     }
17.     printf("\n");
18.     return 0;
19. }
20. void create_list(struct s_list** headp, int* p)
21. {
22.     struct s_list* loc_head = NULL, * tail;
23.     if (p[0] == 0) /* 相当于*p==0 */
24.         ;
25.     else { /* loc_head 指向动态分配的第一个结点 */
26.         loc_head = (struct s_list*)malloc(sizeof(struct s_list))
27.         ;
28.         loc_head->data = *p++; /* 对数据域赋值 */
29.         tail = loc_head; /* tail 指向第一个结点 */
30.         while (*p) { /* tail 所指结点的指针域指向动态创建的结点 */
31.             tail->next = (struct s_list*)malloc(sizeof(struct s_list));
32.             tail = tail->next; /* tail 指向新创建的结点 */
33.             tail->data = *p++; /* 向新创建的结点的数据域赋值 */
34.         }
35.         tail->next = NULL; /* 对指针域赋 NULL 值 */
36.     }
37.     *headp = loc_head; /* 使头指针 headp 指向新创建的链表 */
38.
39. }

```

3) 错误修改后的运行结果:

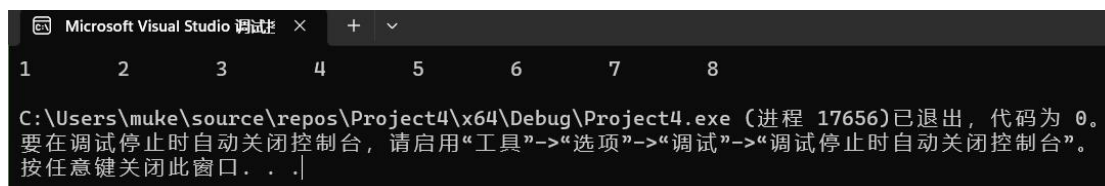


图 7-3 源程序修改替换题错误修改后的程序运行结果示意图

(2) 修改替换 `create_list` 函数, 将其建成一个后进先出的链表, 后进先出链表的头指针始终指向最后创建的结点(链头), 后建结点指向先建结点, 先建结点始终是尾结点。

解答:

1) 更改后的函数如下

```
1. void create_list(struct s_list** headp, int* p) {
2.     struct s_list* loc_head = NULL;
3.
4.     while (*p) {
5.         struct s_list* newx = (struct s_list*)malloc(sizeof(struct s_list));
6.         newx->data = *p++;
7.         newx->next = loc_head; /* 在链表开头插入新节点 */
8.         loc_head = newx;
9.     }
10.
11.     *headp = loc_head; /* 更新 headp 指针, 使其指向最后创建的节点 */
12. }
```

2) 修改后的运行结果



图 7-4 源程序修改替换题函数修改后的程序运行结果示意图

3.程序设计

(1) 本关任务: 用单向链表建立一张班级成绩单, 包括每个学生的学号、姓名、英语、高等数学、普通物理、C 语言程序设计四门课程的成绩。用菜单实现下列功能:

- ① 输入每个学生的各项信息。
- ② 输出每个学生的各项信息。
- ③ 修改指定学生的指定数据项的内容。
- ④ 统计每个同学的平均成绩（保留 2 位小数）。
- ⑤ 输出各位同学的学号、姓名、四门课程的总成绩和平均成绩。

解答：

1) 解题思路：

1. 定义数据结构：首先，定义一个结构体来表示每个学生的信息，包括学号、姓名、英语、高等数学、普通物理和 C 语言程序设计四门课程的成绩。这个结构体可以命名为 `Student`，包含学号（ID）、姓名（name）、英语成绩（english）、高等数学成绩（math）、普通物理成绩（physics）和 C 语言程序设计成绩（programming）。
2. 建立单向链表：创建一个指向 `Student` 结构体的单向链表，每个节点存储一个学生的信息。链表的头指针可以命名为 `head`。
3. 实现菜单功能：

① 输入每个学生的各项信息： 提供一个输入函数，允许用户逐个输入学生的学号、姓名和四门课程的成绩。将每个学生的信息作为一个节点插入链表末尾。

- A. 创建一个函数，比如 `input_student_info()`。
- B. 在函数内，通过用户输入获取学生的学号、姓名以及英语、高等数学、普通物理、C 语言程序设计四门课程的成绩。
- C. 创建一个新的 `Student` 结构体实例，将输入的信息存储在这个实例中。
- D. 将新的学生节点插入链表的末尾

② 输出每个学生的各项信息： 提供一个输出函数，遍历链表并输出每个学生的学号、姓名和四门课程的成绩。

- A. 创建一个函数，比如 `output_student_info()`。
- B. 遍历链表，对于每个节点，输出学生的学号、姓名以及英语、高等数学、普通物理、C 语言程序设计四门课程的成绩。

③ 修改指定学生的指定数据项的内容： 提供一个修改函数，根据用户输入的学号和要修改的数据项，修改相应学生的信息。

- A. 创建一个函数，比如 `modify_student_info()`。
- B. 接受用户输入的学号和要修改的数据项。
- C. 遍历链表，找到对应学号的节点。
- D. 根据用户输入修改相应的数据项，例如，如果用户选择修改英语成绩，则更新节点中的英语成绩。

④ 统计每个同学的平均成绩： 提供一个统计函数，遍历链表，计算每个学生的平均成绩（总成绩除以科目数，保留两位小数）。

- A. 创建一个函数，比如 `calculate_average()`。
- B. 遍历链表，对于每个节点，计算英语、高等数学、普通物理、C 语言程序设计四门课程的平均成绩。
- C. 将平均成绩更新到节点中，保留两位小数。

⑤ 输出各位同学的学号、姓名、四门课程的总成绩和平均成绩： 提供一个输出函数，输出每个学生的学号、姓名、四门课程的总成绩和平均成绩。

- A. 创建一个函数，比如 `output_total_and_average()`。
- B. 遍历链表，对于每个节点，计算总成绩和平均成绩，然后输出学号、姓名、总成绩和平均成绩。

4. 释放内存： 在程序结束时，记得释放链表中每个节点的内存，防止内存泄漏。

5. 用户界面： 使用一个循环，让用户在菜单中选择相应的功能，直到用户选择退出程序。

2) 程序清单

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  #include<string.h>
4.
5.  struct Student //定义学生结构体
6.  {
7.      char ID[10];
8.      char name[20];
9.      int English, math, physics, Cprogramming;
10.     double ave;
```

```

11.  struct Student* next;
12.  };
13.  typedef struct Student stu; //自定义名
14.
15.  stu* head = NULL; //头指针
16.
17.  //输入学生信息
18.  void addStudent()
38.  {
19.      printf("请输入要输入的学生人数:\n");
20.      int n;
21.      scanf("%d", &n);
22.      printf("请输入学生的 ID,姓名, 英语, 数学, 物理, C 语言成绩:\n");
23.      for (int i = 0; i < n; i++) {
24.          stu* student = (stu*)malloc(sizeof(stu)); //分配空间
25.          scanf("%s", student->ID, 10);
26.          scanf("%s", student->name, 20);
27.          scanf("%d %d %d %d",
28.              &student->English, &student->math,
29.              &student->physics, &student->Cprogramming); //输入成
绩
30.
31.          student->next = NULL; //设置为空
32.          if (head == NULL)
33.          {
34.              head = student; //head 指向第一位
35.          }
36.          else
37.          {
38.              stu* current = head;
39.              while (current->next != NULL)
40.              {
41.                  current = current->next; //向下查找到结尾
42.              }
43.              current->next = student; //让上一个结构体指向本个
44.          }
45.
46.      }
47.      printf("完成了%d 位同学的成绩输入。 \n", n);
48.  }
49.  //展示学生信息
50.  void display1() {
51.
52.      struct Student* current = head;

```

```

53.   printf("学生信息如下:\n");
54.   while (current != NULL) {
55.       printf("%-10s\t", current->ID);
56.       printf("%-10s\t", current->name);
57.       printf("%-3d%-3d%-3d%-3d\n", current->English,current->
        math,current->physics
58.           ,current->Cprogramming);
59.       current = current->next;
60.   }
61. }
62. void display2()
63. {
64.     struct Student* current = head;
65.
66.     while (current != NULL) {
67.         printf("%s\t", current->ID);
68.         printf("%s\t", current->name);
69.         printf("%d\t", current->Cprogramming + current->English
        + current->physics
70.             + current->math);
71.         printf("%.2f\n", current->ave);
72.         current = current->next;
73.     }
74. }
75. void sta()
76. {
77.     struct Student* current = head;
78.     while (current != NULL) {
79.         current->ave = (current->English + current->math +
80.             current->physics + current->Cprogramming) / 4.0;
81.         current = current->next;
82.     }
83.
84. }
85. //排序算法
86. void sortScores() {
87.     // 如果链表为空或只有一个节点, 无需排序
88.     if (head == NULL || head->next == NULL) {
89.         return;
90.     }
91.
92.     int swap = 1;        // 用于检测是否发生交换
93.     stu* temp = NULL;    // 用于交换节点
94.     stu* end = NULL;     // 指向已排序部分的末尾

```

```

95.
96. while (swap) { // 若没有发生交换, 即 swap 为 0, 则退出循环
97.     swap = 0;           // 在每次外部循环开始时重置交换标志
98.     stu* current = head; // 从链表头开始遍历
99.
100. while (current->next != end) {
101.     stu* next = current->next;
102.     if (current->ave > next->ave)
103.     {
104.         // 交换节点
105.         if (current == head) {
106.             head = next; // 重置头指针
107.         }
108.         else {
109.             temp->next = next; // temp 为 prev, 更改指向
110.         }
111.
112.         current->next = next->next; // 交换
113.         next->next = current;
114.         temp = next; // temp 向右移动
115.         swap = 1; // 标记发生了交换
116.     }
117.     else {
118.         temp = current;
119.         current = current->next;
120.     }
121. }
122. end = current; // 每轮大循环结束时, 已排序部分的末尾为当前
    节点
123. }
124. }
125. // 修改学生信息
126. void modify() {
127.     char targetID[10];
128.     int Type;
129.     char Content[20];
130.     printf("修改项目:\n");
131.     printf("0.姓名\n1.英语成绩\n2.数学成绩\n3.物理成绩\n4.C 语言
        成绩\n");
132.     printf("请输入要修改的学生的学号, 修改项目, 修改后的内容: ");
133.     scanf("%s%d%s", targetID, &Type, Content);
134.
135.     stu* current = head;
136.     while (current != NULL) {

```

```

137.  if (strcmp(current->ID, targetID) == 0) //比较是否相等
138.  {
139.      switch (Type) {
140.          case 0:
141.              strcpy(current->name, Content);
142.              break;
143.          case 1:
144.              current->English = atoi(Content);
145.              break;
146.          case 2:
147.              current->math = atoi(Content); //atoi 将字符串转化为整型
148.              break;
149.          case 3:
150.              current->physics = atoi(Content);
151.              break;
152.          case 4:
153.              current->Cprogramming = atoi(Content);
154.              break;
155.          default:
156.              printf("无效的修改类型\n");
157.              return;
158.      }
159.      current->ave = (double)(current->English + current->ma
160.          th
161.          + current->physics + current->Cprogramming) / 4.0;
162.      printf("%s ", current->ID);
163.      printf("%s ", current->name);
164.      printf("%d %d %d %d\n", current->English, current->mat
165.          h,
166.          current->physics, current->Cprogramming);
167.      return;
168.  }
169.  }
170. }
171. void menu()
172. {
173.     printf("请输入选项\t\n");
174.     printf("1.输入每个学生的各项信息\n");
175.     printf("2.输出每个学生的各项信息\n");
176.     printf("3.修改指定学生的指定数据项的内容\n");

```

```

177. printf("4.统计每个同学的平均成绩\n");
178. printf("5.输出各位同学的学号、姓名、四门课程的总成绩和平均成绩\n");
179. printf("6.按照平均成绩进行升序排序\n");
180. printf("0.退出程序\n");
181. }
182. int main()
183. {
184.     menu();
185.     int choice = -1;
186.     while (choice != 0)
187.     {
188.         scanf("%d", &choice);
189.         switch (choice)
190.         {
191.             case 1:addStudent();
192.             break;
193.             case 2:display1();
194.             break;
195.             case 3:modify();
196.             break;
197.             case 4:sta();
198.             break;
199.             case 5:display2();
200.             break;
201.             case 6:sortScores();
202.             break;
203.             case 0:exit(0);
204.
205.             default:printf("无效输入\n");
206.
207.         }
208.     }
209.     //释放内存
210.     stu* current = head;
211.     while (current != NULL) {
212.         stu* next = current->next;
213.         free(current);
214.         current = next;
215.     }
216.
217.     return 0;
218. }

```


3) 测试

(a) 测试数据:

表 7-1 编程题 1 的测试数据

测试用例	程序输入	理论结果
用例 1	1 5 2021001 Jack 90 92 87 95 2021002 Mike 85 70 75 90 2021003 Joe 77 86 90 75 2021004 Andy 95 97 92 95 2021005 Rose 90 87 88 89	完成了 5 位同学的成绩输入。
用例 2	2	学生信息如下: 2021001 Jack 90 92 87 95 2021002 Mike 85 70 75 90 2021003 Joe 77 86 90 75 2021004 Andy 95 97 92 95 2021005 Rose 90 87 88 89
用例 3	3 2021001 1 100	2021001 Jack 100 92 87 95
用例 4	4	已完成平均分统计
用例 5	5	2021001 Jack 374 93.50 2021002 Mike 320 80.00 2021003 Joe 328 82.00 2021004 Andy 379 94.75 2021005 Rose 354 88.50

(b) 对应测试测试用例的运行结果如图 7-5, 7-6 所示。

```

请输入选项
1. 输入每个学生的各项信息
2. 输出每个学生的各项信息
3. 修改指定学生的指定数据项的内容
4. 统计每个同学的平均成绩
5. 输出各位同学的学号、姓名、四门课程的总成绩和平均成绩
6. 按照平均成绩进行升序排序
0. 退出程序
1
请输入要输入的学生人数:
5
请输入学生的ID, 姓名, 英语, 数学, 物理, C语言成绩:
2021001 Jack 90 92 87 95
2021002 Mike 85 70 75 90 2021003
Joe 77 86 90 75
2021004 Andy 95 97 92 95 2021005
Rose 90 87 88 89
完成了5位同学的成绩输入。
2
学生信息如下:
2021001      Jack      90 92 87 95
2021002      Mike      85 70 75 90
2021003      Joe       77 86 90 75
2021004      Andy      95 97 92 95
2021005      Rose      90 87 88 89
    
```

图 7-5 编程题 1 的测试用例 1-2 的运行结果

```

3 2021001 1 100
修改项目:
0. 姓名
1. 英语成绩
2. 数学成绩
3. 物理成绩
4. C语言成绩
请输入要修改的学生的学号, 修改项目, 修改后的内容: 2021001 Jack 100 92 87 95
4
5
2021001 Jack 374 93.50
2021002 Mike 320 80.00
2021003 Joe 328 82.00
2021004 Andy 379 94.75
2021005 Rose 354 88.50
    
```

图 7-6 编程题 1 的测试用例 3-5 的运行结果

(2) 本关任务: 对程序设计题第(1)题的程序, ⑥增加按照平均成绩进行升序排序的函数, 写出用交换结点数据域的方法升序排序的函数, 排序可用选择法或冒泡法。

解答:

1) 解题思路:

利用冒泡法和中间变量 t, 交换结点数据域

2) 程序清单

```
1. void sortScores() {
```

```
2. // 如果链表为空或只有一个节点，无需排序
3. if (head == NULL || head->next == NULL) {
4.     return;
5. }
6.
7. int swap = 1; // 用于检测是否发生交换
8. stu* end = NULL; // 指向已排序部分的末尾
9.
10. while (swap) { // 若没有发生交换，即 swap 为 0，则退出循环
11.     swap = 0; // 在每次外部循环开始时重置交换标志
12.     stu* current = head; // 从链表头开始遍历
13.
14.     while (current->next != end) {
15.         stu* next = current->next;
16.         if (current->ave > next->ave) {
17.             // 交换数据域
18.             int tempAve = current->ave;
19.             current->ave = next->ave;
20.             next->ave = tempAve;
21.
22.             // 如果有其他数据域，也需要类似的交换
23.
24.             // 标记发生了交换
25.             swap = 1;
26.         }
27.         current = current->next;
28.     }
29.     end = current; // 每轮大循环结束时，已排序部分的末尾为当前
        节点
30. }
31. }
```

3) 测试
(a) 测试数据:

表 7-2 编程题 2 的测试数据

测试用例	程序输入	理论结果		
用例 1	6 5	2021002 Mike	320	80.00
		2021003 Joe	328	82.00
		2021005 Rose	354	88.50
		2021001 Jack	374	93.50
		2021004 Andy	379	94.75

(b) 对应测试用例 1 的运行结果如图 7-7 所示。

```
6
5
2021002 Mike      320      80.00
2021003 Joe       328      82.00
2021005 Rose      354      88.50
2021001 Jack      374      93.50
2021004 Andy      379      94.75
```

图 7-7 编程题 2 的测试用例的运行结果
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。

7.3 实验小结

本次实验主要考察对各种操作符优先级，运算顺序，结构体和链表的使用的理解。通过实验，我熟悉了结构的说明和引用、结构的指针、结构数组、以及函数中使用结构的方法。了解动态储存分配函数的用法，掌握自引用结构，单向链表的创建、遍历、结点的增删、查找等操作。实验过程中，我也遇到了一些问题，以下是我的经验和总结。

1. **善用调试：**在程序改错题中，我没有发现传入的是一级指针，函数形参的改变不会影响原指针的值，通过调试，我发现了主函数中 p 是个空指针，因此程序没有任何输出，从而发现了问题所在。
2. **运算优先级：**通过第一道题，我总结出，同一优先级的运算符，运算次序由结合方向所决定。简单记就是：！ > 算术运算符 > 关系运算符 > && > || > 赋值运算符。
3. **大型程序编写：**程序设计题中，我们需要做一个具有六个功能的学生管理系统。这既需要我们编写多个函数实现相应功能，还需要我们设计格式，提示语，注意各个函数中链表的内存分配，排序等是否存在问题，这对提升我们的编程能力十分重要。

实验 8 文件操作实验

8.1 实验目的

- (1) 熟悉文本文件和二进制文件在磁盘中的存储方式;
- (2) 熟练掌握流式文件的读写方法。

8.2 实验题目及要求

1. 文件类型的程序验证题

设有程序:

```
1.  #include <stdio.h>
2.  int main(void)
3.  {
4.      short a=0x253f,b=0x7b7d;
5.      char ch;
6.      FILE *fp1,*fp2;
7.      fp1=fopen("d:\\abc1.bin","wb+");
8.      fp2=fopen("d:\\abc2.txt","w+");
9.      fwrite(&a,sizeof(short),1,fp1);
10.     fwrite(&b,sizeof(short),1,fp1);
11.     fprintf(fp2,"%hx %hx",a,b);
12.
13.     rewind(fp1); rewind(fp2);
14.     while((ch = fgetc(fp1)) != EOF)
15.         putchar(ch);
16.     putchar('\n');
17.
18.     while((ch = fgetc(fp2)) != EOF)
19.         putchar(ch);
20.     putchar('\n');
21.
22.     fclose(fp1);
23.     fclose(fp2);
24.     return 0;
25. }
```

- (1) 请思考程序的输出结果, 然后通过上机运行来加以验证。

解答:

1) 第一个输出是从"abc1.bin"文件中读取的字符, 因为写入时使用了二进制写入, 所以直接输出的是文件中的原始二进制数据。由于以字符形式输出, 依次输出一个字节 ASCII 码值对应的字符因此输出为?(对应 ASCII 码值 63)%(对应 ASCII 码值 37), b 同理, 对应}和{

2) 第二个输出是从"abc2.txt"文件中读取的字符, 因为写入时使用了十六进制格式, 所以输出的是 a 和 b 的十六进制表示, 即 253f 7b7d

3) 因此, 程序理论输出为:

```
?%}{
253f 7b7d
```

4) 运行的结果如下所示:

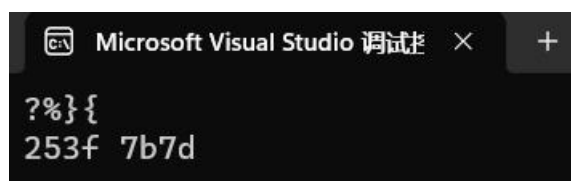


图 8-1 程序验证题(1)的运行结果

(2) 将两处 sizeof(short)均改为 sizeof(char)结果有什么不同, 为什么?

解答:

1) 由于 sizeof(char)大小为 1 个字节, 因此只写入 a, b 的低 8 位, 对应输出?}

2) 运行的结果如下所示:

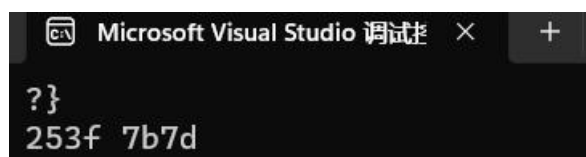


图 8-2 程序验证题(2)修改后的运行结果

(3) 将 fprintf(fp2,"%hx %hx",a,b) 改为 fprintf(fp2,"%d %d",a,b)结果有什么不同。

解答:

1) 由于以%d 的形式写入 a, b, 而 a, b 对应的十进制数字是 9535 和 31613

2) 因此, 程序将输出:

```
?%}{
```

9535 31613

3) 运行的结果如下所示:



图 8-3 程序验证题(3)修改后的运行结果

2. 源程序修改替换题

将指定的文本文件内容在屏幕上显示出来，命令行的格式为：

type filename

- (1) 源程序中存在什么样的逻辑错误（先观察执行结果）？对程序进行修改、调试，使之能够正确完成指定任务。

```

1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  int main(int argc, char* argv[])
4.  {
5.      char ch;
6.      FILE *fp;
7.      if(argc!=2){
8.          printf("Arguments error!\n");
9.          exit(-1);
10.     }
11.     if((fp=fopen(argv[1],"r"))==NULL){          /* fp 指
        向 filename */
12.         printf("Can't open %s file!\n",argv[1]);
13.         exit(-1);
14.     }
15.
16.     while(ch=fgetc(fp)!=EOF)                    /* 从filename 中读字
        符 */
17.         putchar(ch);                            /* 向显示器中写字符 */
18.     fclose(fp);                                /* 关闭filename */
19.     return 0;
20. }
```

解答：

- 1) 第 16 行出错，这里的问题在于，赋值操作符 = 的优先级低于比较操作符 !=，

因此在这个语句中，先执行了 `ch=fgetc(fp)`，然后将结果与 `EOF` 比较。这导致 `ch` 实际上被赋值为比较的结果，而不是 `fgetc(fp)` 的返回值。为了修复这个问题，需要加上括号，确保赋值操作优先执行：`while((ch=fgetc(fp))!=EOF)`

2) 错误修改后运行结果：

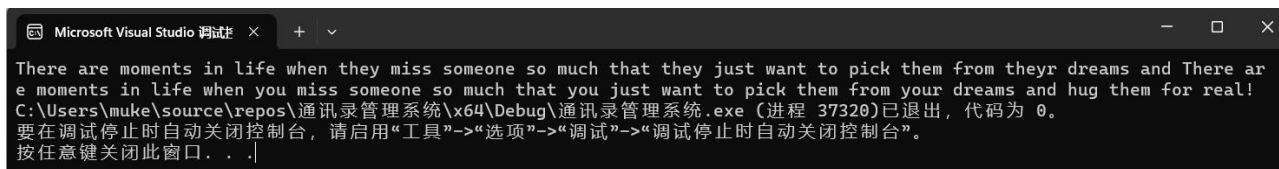


图 8-4 源程序修改替换题错误修改后的运行结果

(2) 用输入输出重定向 `freopen` 改写 `main` 函数。

解答：

1) 修改替换后的源程序如下：

```

1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  int main(int argc , char* argv[])
4.  {
5.      char ch;
6.      FILE *fp;
7.      if(argc!=2)
8.      {
9.          printf("Arguments error!\n");
10.         exit(-1);
11.     }
12.     if((freopen(argv[1],"r",stdin))==NULL)
13.     {          /* fp 指向 filename */
14.         printf("Can't open %s file!\n",argv[1]);
15.         exit(-1);
16.     }
17.
18.     while((ch=getchar())!=EOF)          /* 从 filename 中读字
        符 */
19.         putchar(ch);                    /* 向显示器中写字符 */
20.     return 0;
21. }
```

2) 修改替换后运行结果：

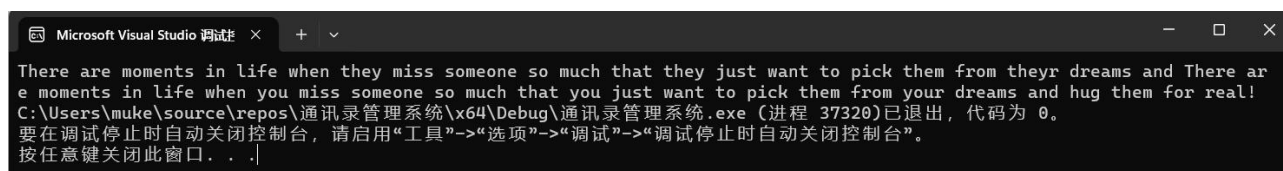


图 8-5 源程序修改替换题修改替换后的运行结果

3. 程序设计

(1) 编写一个程序，用给定的字符串替换文件中的目标字符串，并显示输出替换的个数。

注意：读取的文件路径请使用 `experiment/src/step8/source.txt`

若文件为

``There are moments in life when you miss someone so much that you just want to pick them from your dreams and hug them for real!``

样例输入：`you they`

样例输出：

``3``

``There are moments in life when they miss someone so much that they just want to pick them from theyr dreams and hug them for real!``

解答：

1) 解题思路：

1. 文件读取： 打开指定路径的文件（`"experiment/src/step8/source.txt"`）以读取方式，使用 `fread` 将文件内容读取到字符数组 `str1` 中。
2. 输入目标字符串和替换字符串： 通过 `scanf` 从用户获取两个字符串，分别存储在 `str2` 和 `str3` 中，这两个字符串分别代表目标字符串和替换字符串。
3. 查找与替换： 使用 `strstr` 函数查找目标字符串 `str2` 在源文本 `str1` 中的位置。若找到了目标字符串，就调用自定义函数 `str_replace` 进行替换。`str_replace` 函数会根据目标字符串和替换字符串的长度关系进行适当的移动和替换操作。
4. 替换计数： 在找到并替换目标字符串的过程中，使用变量 `count` 记录替换的次数。
5. 输出结果： 输出替换的总次数和修改后的文本内容。

2) 程序清单

```
1. #include <string.h>
```

```

2.  #include <stdio.h>
3.
4.  // 自定义函数，用于替换字符串
5.  void str_replace(char* position, int len_target, char* re
    placement)
6.  {
7.      int len_replace;
8.      char* tmp;
9.      len_replace = strlen(replacement);
10.     // 替换字符串比目标字符串短，往前移动
11.     if (len_replace < len_target)
12.     {
13.         tmp = position + len_target;
14.         while (*tmp)
15.         {
16.             *(tmp - (len_target - len_replace)) = *tmp; // len_tar
                get - len_replace 是移动的距离
17.             tmp++; // 不断左
                移
18.         }
19.         *(tmp - (len_target - len_replace)) = *tmp; // 移动终止
                符号
20.     }
21.     else
22.         // 替换比目标长，往后移动
23.         if (len_replace > len_target)
24.         {
25.             tmp = position;
26.             while (*tmp) tmp++;
27.             while (tmp >= position + len_target)
28.             {
29.                 *(tmp + (len_replace - len_target)) = *tmp;
30.                 tmp--;
31.             }
32.         }
33.         strncpy(position, replacement, len_replace);
34.     }
35.
36. int main()
37. {
38.     char str1[1000] = { 0 };
39.     char str2[10], str3[10];
40.     int count = 0;
41.     char* p;

```

```

42.
43. // 从用户输入获取目标字符串和替换字符串
44. scanf("%s", str2);
45. scanf("%s", str3);
46.
47. // 读取文本文件
48. FILE* fp = fopen("experiment/src/step8/source.txt", "r")
    ;
49. fread(str1, 1, 1000, fp);
50.
51. // 开始查找字符串 str2
52. p = strstr(str1, str2);
53. while (p)
54. {
55.     count++;
56.     // 每找到一个str2, 就用str3 来替换
57.     str_replace(p, strlen(str2), str3);
58.     p = p + strlen(str3);
59.     p = strstr(p, str2);
60. }
61.
62. // 输出替换的总次数和修改后的文本内容
63. printf("%d\n", count);
64. printf("%s", str1);
65.
66. // 关闭文件
67. fclose(fp);
68.
69. return 0;
70. }

```

3) 测试

(a) 测试数据:

表 8-1 编程题 1 的测试数据

测试用例	文件内容	程序输入	理论结果
用例 1	There are moments in life when you miss someone so much that you just want to pick them from your dreams and hug them for real!	you they	3 There are moments in life when they miss someone so much that they just want to pick them from theyr dreams and hug them for real!

用例 2	There are moments in life when you miss someone so much that you just want to pick them from your dreams and hug them for real!	express y	0 There are moments in life when you miss someone so much that you just want to pick them from your dreams and hug them for real!
------	---------------------------------------------------------------------------------------------------------------------------------	-----------	-----------------------------------------------------------------------------------------------------------------------------------

(b) 对应测试用例 1 的运行结果如图 8-6 所示。

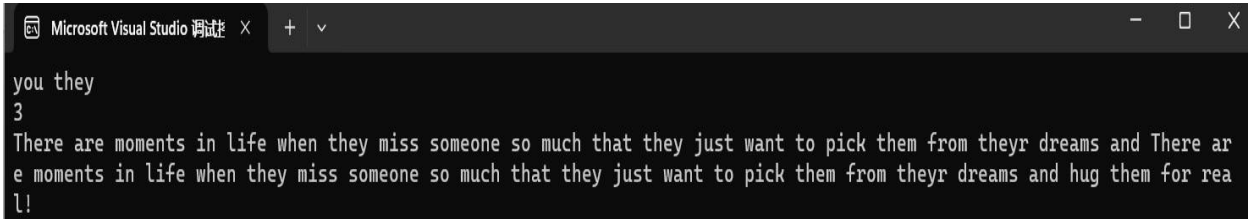


图 8-6 编程题 1 的测试用例一的运行结果

对应测试用例 2 的运行结果如图 8-7 所示。

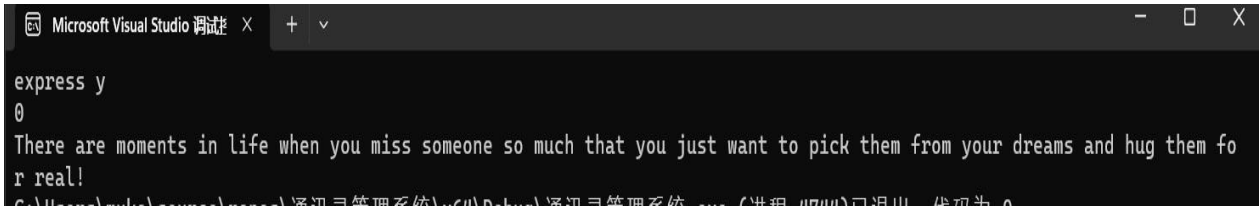


图 8-7 编程题 1 的测试用例二的运行结果

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。

8.3 实验小结

通过实验，我熟悉了文本文件和二进制文件在磁盘中的存储方式，熟练掌握流式文件的读写方法。我们首先学习了文本文件和二进制文件在磁盘中的存储方式。文本文件以可读的字符形式存储，而二进制文件以字节的形式存储。理解这两者的不同有助于更好地选择适当的文件类型和读写形式。另外，我们学习了流式文件的读写方法，这包括使用标准库提供的函数（如 fopen、fclose、fread、fwrite 等）来打开、关闭、读取和写入文件。这些函数提供了方便的接口，使得文件的操作变得相对简单。以下是我实验过程中的体会和经验。

1. **注意路径细节:**当你在 C 语言中表示文件路径时, 路径通常包含反斜杠, 例如: "C:\Users\Username\Documents\file.txt"。但是, 单个反斜杠在 C 语言中会被解释为转义字符的开始, 而不是普通的字符。为了表示一个普通的反斜杠, 你需要使用两个反斜杠, 即: "C:\\Users\\Username\\Documents\\file.txt"。
2. 这样做是为了防止编译器将反斜杠后面的字符解释为转义字符, 而保持反斜杠的字面意义。在字符串中使用两个反斜杠就是告诉编译器要插入一个普通的反斜杠, 而不是一个转义字符。
3. **养成良好的代码检查习惯:**在涉及文件操作的程序编写时, 我们需要记得利用 if 语句判断 FILE 指针是否成功指向我们所需的问题, 同时在使用结束后是否 fclose, 这是一个良好的习惯, 否则可能会造成潜在的问题。
4. **理解函数细节:**在程序设计题中, 我起初遇到了一个问题, 即利用 strstr 进行查找替换时, 若替换字符数与原字符数不相同, 则会造成空缺和多余。通过学习和思考, 我明白应该自定义一个函数进行字符串中元素的移动, 在这里, 终止符成为我们判断终点的重要工具, 同时我们也要记得重新设置终止符。