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： 计算机类 202011 班

学　　号： U202015084

姓　　名： 张文浩

指导教师： 卢萍

报告日期： 2021.6.5

计算机科学与技术学院

目 录

1 实验 6 指针

1.1 程序改错与跟踪调试

在下面给出的源程序中，函数 strcopy(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。(错误处以下划线标出）

```
#include<stdio.h>
char* strcopy(char*, const char*);
int main()
{
    char *s1,*s2,*s3;
    scanf("%s", s2);
    strcopy(s1, s2);
    printf("%s\n", s1);
    scanf("%s", s2);
    strcopy(s3, s2);
    printf("%s\n", s3);
    return 0;
}
char* strcopy(char* t, const char* s)
{
    while (*t++ = *s++);
    return t;
}
```

错因：s1,s2,s3 为悬挂指针，没有指向具体的地址，不能用 scanf 对其赋值或者对其直接写入值。

修改：将 char *s1,*s2,*s3 改为 char s1[100],s2[100],s3[100]。

1.2 程序完善与修改替换

1. 字符串升序排序

下面程序函数 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，存入通过 malloc 动态分配的存储空间，然后调用 strsort 对这 n 个字符串按字典序升序排序。

（1）程序完善（填空处用下划线标出）

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void strsort(char* s[], int size)
{
    char* temp;
    int i, j;
    for(i=0;i<size-1;i++)
        for(j=0;j<size-i-1;j++)
            if (strcmp(s[j],s[j+1])>0)
            {
                temp = s[j];
                s[j] = s[j + 1];
                s[j + 1] = temp;
            }
}
int main()
{
    int i,n;
    char** s, t[50];
    scanf("%d", &n);
    s = (char**)malloc(sizeof(char*) * n);
    for (i = 0; i < n; i++)
    {
        scanf("%s", t);
        s[i] = (char*)malloc(strlen(t) + 1);
        strcpy(s[i], t);
```

```
    }
    strsort(s, n);
    for (i = 0; i < n; i++)
    {
        puts(s[i]);
        free(s[i]);
    }
    free(s);
    return 0;
}
```

（2）程序修改替换：用二级指针形参重写 strsort 函数。并且该函数体的任何位置都不允许使用下标引用。

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void strsort(char **s, int size)
{
    char* temp;
    int i, j;
    for(i=0;i<size-1;i++)
        for(j=0;j<size-i-1;j++)
            if (strcmp(*(s+j),*(s+j+1) )>0)
            {
                temp = *(s+j);
                *(s+j) = *(s+j+1);
                *(s+j+1) = temp;
            }
}
int main()
{
```

```
    int i,n;
    char** s, t[50];
    scanf("%d", &n);
    s = (char**)malloc(sizeof(char*) * n);
    for (i = 0; i < n; i++)
    {
        scanf("%s", t);
        s[i] = (char*)malloc(strlen(t) + 1);
        strcpy(s[i], t);
    }
    strsort(s, n);
    for (i = 0; i < n; i++)
    {
        puts(s[i]);
        free(s[i]);
    }
    free(s);
    return 0;
}
```

2. 字符串操作

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）程序完善（填空处用下划线标出）

```
#include<stdio.h>
#include<string.h>
int main()
{
    char* (*p)(char*, const char*)=NULL;
    char a[80], b[80], * result;
    int choice;
```

```
    while (1)
    {
        do
        {
            scanf("%d", &choice);
        } while (choice < 1 || choice>4);
        switch (choice)
        {
        case 1:
            p = strcpy;
            break;
        case 2:
            p = strcat;
            break;
        case 3:
            p = strtok;
            break;
        case 4:
            goto down;
        }
        getchar();
        gets(a);
       gets(b);
        result = p(a, b);
        printf("%s\n", result);
    }
down:
    return 0;
}
```

（2）程序修改替换：通过转移表实现多分支函数处理。

```
#include<stdio.h>
#include<string.h>
int main()
{
    char* (*p[4])(char*, const char*) = { strcpy,strcat,strtok};
    char a[80], b[80], * result;
    int choice;
    while (1)
    {
        do
        {
            scanf("%d", &choice);
        } while (choice < 1 || choice>4);
        if (choice == 4) goto down;
        getchar();
        gets(a);
        gets(b);
        result = p[choice-1](a, b);
        printf("%s\n", result);
    }
down:
    return 0；
}
```

1.3 程序设计

1. 通过指针取出字节

解题思路：将一 char 型指针指向变量，通过循环和位运算取出每个字节的高 4 位和低 4 位。

算法步骤：用一 char 类型指针指向强制转换为 char 型的某整型变量的地址；

通过指针的加法运算与循环语句实现移动指针指向从高字节到低字节的 4 个字节；对每个字节，将此时指针取内容后与 0xf0 进行按位与运算并右移 4 位，取出高 4 位，指针取内容与 0x0f 进行按位与运算，取出低 4 位。

```
#include<stdio.h>
int main()
{
    int n;
    scanf("%d", &n);
    char* p=(char *)&n; //
    for (int i = 3; i >=0 ; i--) /*从 3 开始，实现从高字节到低字节的移动*/
    {
        printf("%x %x", (*(p+i)&0xf0)>>4,*(p+i)&0x0f);
       /*前者取高 4 位，后者取低 4 位*/
        if (i != 0) printf(" ");
    }
    return 0;
}
```

2.删除重复元素：

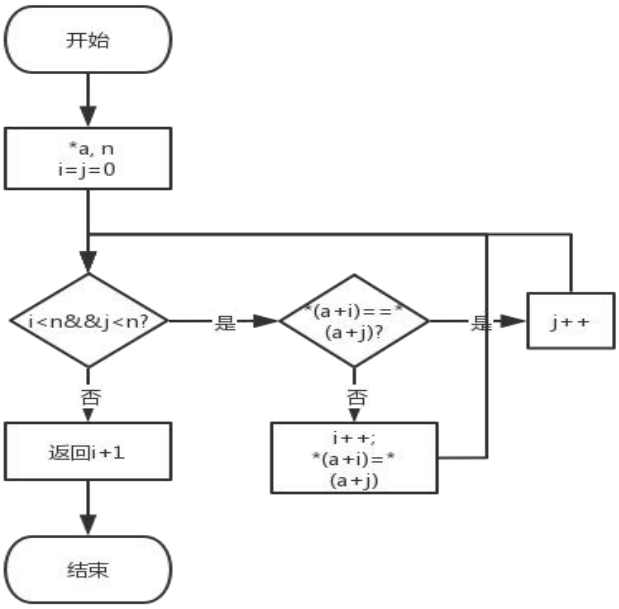

图 1-1 程序设计题 2 流程图

解题思路：设置两个指针 i，j，当两指针指向相同元素时移动 j，指向不同时将 i 后移一位，然后将 j 的指向值赋值给 i 指向的值，这样就实现了重复的数值的覆盖，最后返回 i+1，即为非重复元素个数。

代码：

```
#include<stdio.h>
int RemoveSame(int* a, int n);
int main()
{
    int *a,n;
     scanf("%d",&n);
     a=(int *)malloc(sizeof(int)*n);
     for(int i=0;i<n;i++)
          scanf("%d",a+i);
    int k=RemoveSame(a, n);
    for (int i = 0; i < k; i++)
     {
        printf("%d", a[i]);
          if(i!=k-1) printf(" ");
     }
     printf("\n%d",k);
     free(a);
    return 0;
}
int RemoveSame(int* a, int n)
{
    int i, j;
    for (i = j = 0; i < n && j < n;)
    {
        if (*(a+i) != *(a + j))
```

```
            {
                i++;
               /*i 指向的元素的后一个是第一个重复元素，用 j 指向的值覆盖*/
                *(a + i) = *(a + j);
            }
            else j++; /*j 后移，直到找到第一个不重复元素*/
        }
        return i+1; /*i 是数组下标，i+1 后才是非重复元素个数*/
    }
```

3.旋转图像

解题思路：将矩阵压缩为一维数组，通过指针操作打印出正确位置的元素。

算法步骤：双层循环，第一层从最后一列开始遍历各列，第二层从第一行开始遍历各行，以此实现将原矩阵逆时针旋转 90°

代码：

```
#include<stdio.h>
#include<stdlib.h>
void revolve(int* a,int n,int m);
int main()
{
    int* a, n, m;
    scanf("%d%d", &n, &m);
    a = (int*)malloc(sizeof(int) * n * m);
    for (int i = 0; i < n*m; i++)
        scanf("%d", &a[i]);
       /*二维数组压缩为一维数组，作为函数参数比较简便*/
    revolve(&a[0],n,m);
    free(a);
    return 0;
}
void revolve(int* a,int n,int m)
```

```
{
    for (int j = m - 1; j >= 0; j--) /*从最后一列开始遍历*/
    {
        for (int i = 0; i < n; i++) /*从第一行开始遍历*/
        {
            printf("%d", *(a+i*m+j)); /*取出第 i 行第 j 列的元素*/
            if (i != n - 1) printf(" ");
        }
        printf("\n");
    }
}
```

4.命令行实现对 n 个整数排序

解题思路：通过命令行输入整数个数和排序依据，然后排序。

算法步骤：判断输入的命令里面是否含有-d，若含有，就按递减顺序进行冒泡排序，否则按递增顺序。

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void sort(int* a, int n, int flag);
int main(int argc, char* argv[]) {
        int *a,n=atoi(argv[1]), flag; /*flag 是决定排序方式的变量*/
          if(argc==3)
              flag=strcmp("-d",argv[2]); /*若是-d，则 flag=0*/
          else /*此时一定没有参数-d*/
              flag=1;
        a = (int*)malloc(sizeof(int) * n);
        for (int i = 0; i < n; i++)
            scanf("%d", &a[i]);
        sort(a, n, flag);
```

```
    for (int i = 0; i < n; i++)
    {
        printf("%d", a[i]);
        if (i != n - 1) printf(" ");
    }
    free(a);
    return 0;

}
void sort(int* a, int n, int flag) /*简单的冒泡排序*/
{
    for(int i=0;i<n-1;i++)
    for (int j = 0; j < n - i - 1; j++)
    {
        if (flag != 0)
        {
            if (a[j] > a[j + 1])
            {
                int temp;
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
        else
        {
            if (a[j] < a[j + 1])
            {
                int temp;
                temp = a[j];
```

```
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}
```

5. 删除字串

解题思路：不断对原字符串进行 strstr 操作，如果含有要删除的字符串时就利用 strcpy 将字符串删除。

流程图：

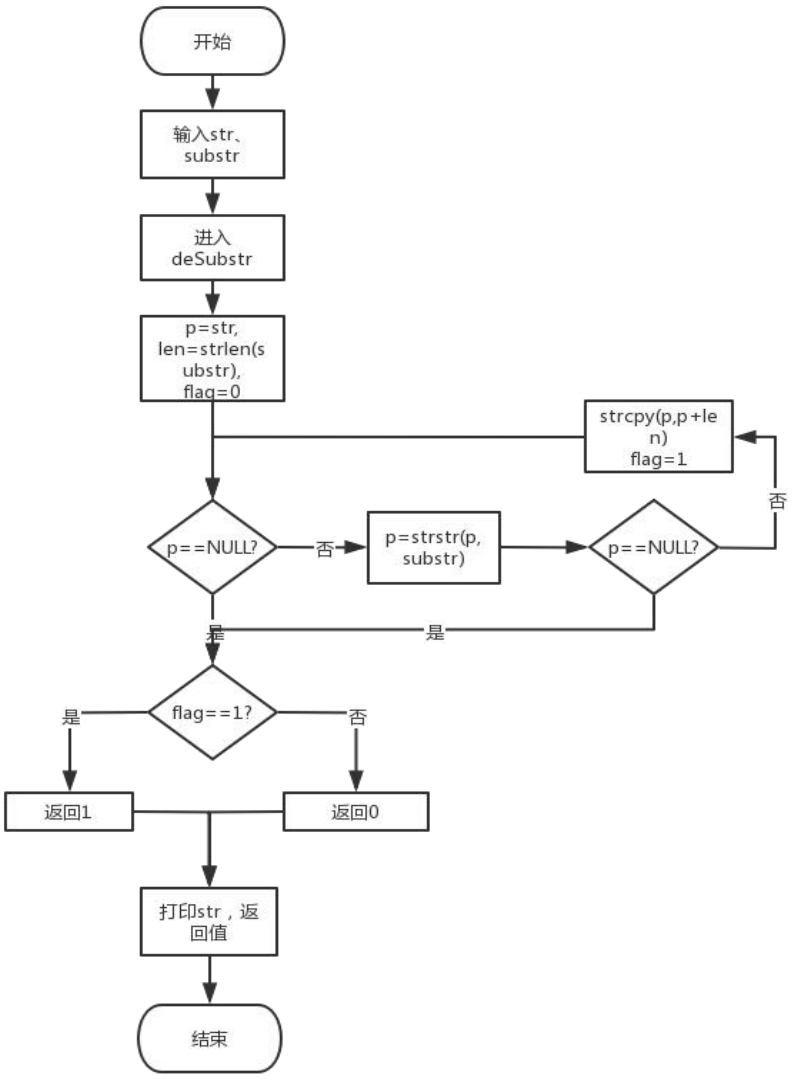

图 1-2 程序设计题 5 流程图

代码：

```
#include<string.h>
#include<stdlib.h>
int deSubstr(char *str,char *substr);
int main() {
    char* str = (char*)malloc(sizeof(char) * 100);
    char* substr = (char*)malloc(sizeof(char) * 100);
    gets(str);
    gets(substr);
    int i = deSubstr(str, substr);
    printf("%s\n%d", str, i);
    return 0;
}
int deSubstr(char* str, char* substr)
{
    int len = strlen(substr),flag=0; /*flag 用来标记是否删除过*/
    char *p=str;
    while (p)
    {
        p = strstr(p, substr);   /*每次都是从当前位置往后找字串*/
        if (p == NULL) break;
        strcpy(p, p + len); /*通过复制操作删除字串*/
        flag = 1;
    }
    if (flag == 1) return 1;
    else return 0;
}
```

6. 非负整数积

解题思路：逆向储存两个数字，那么对于第一个数字的第 i 位和第二个数字

的第 j 位，他们只会对结果的 i+j-1 位有贡献，然后处理进位和前导 0

算法步骤：模拟竖式运算。逆向储存两个数字的每一位，然后两层循环遍历两个数字，结果的 i+j-1 位每次都加上第一个的第 i 位乘以第二个的第 j 位。然后遍历结果的每一位，对结果的每一位，其下一位都等于原来的数加上该位除以 10，相当于进位，然后该位对 10 取余表示进位后剩下的数。对结果，如果他最后一位为 0，就让其长度减一，依次消除前导 0，直到其最后一位不为 0，然后逆序输出结果的每一位。

代码：

```
#include<stdio.h>
#include<string.h>
char num1[201], num2[201];
int a[210] = { 0 }, b[210] = { 0 }, c[210] = { 0 };
int main()
{
    int l1, l2;
    scanf("%s%s", num1,num2);
    l1 = strlen(num1);
    l2 = strlen(num2);
    /*反向储存各位数字*/
    for (int i = 1; i <= l1; i++)
        *(a+i) = *(num1+l1-i) - '0';
    for (int i = 1; i <= l2; i++)
        *(b+i) = *(num2+l2-i) - '0';
    /*模拟竖式运算*/
    for (int i = 1; i <= l1; i++)
        for (int j = 1; j <= l2; j++)
            *(c+i+j-1) += (*(a+i)) * (*(b+j)) ;
    /*进位*/
    for (int i = 1; i <= l1 + l2; i++)
    {
```

```
            *(c+i+1) += *(c+i) / 10;
            *(c+i) %= 10;
        }
        int len = l1 + l2;
        /*除去前导 0*/
        while (len > 1 && *(c+len) == 0)
            len--;
        /*反向输出*/
        for (int i = len; i >= 1; i--)
            printf("%d", *(c+i));
        return 0;
    }
```

7. 函数调度

算法步骤：创建一个指向八个任务函数的函数指针数组，根据输入的数字序列，依次调用对应序号的函数。

代码：

```
#include<stdio.h>
#include<string.h>
void (*p[8])(void);
void task0();
void task1();
void task2();
void task3();
void task4();
void task5();
void task6();
void task7();
void scheduler();
void execute(void (*p[])(void),int n);
int main()
```

```
{
    scheduler();
    return 0;
}
void task0()
{
    printf("task0 is called!\n");
}
void task1()
{
    printf("task1 is called!\n");
}
void task2()
{
    printf("task2 is called!\n");
}
void task3()
{
    printf("task3 is called!\n");
}
void task4()
{
    printf("task4 is called!\n");
}
void task5()
{
    printf("task5 is called!\n");
}
void task6()
{
```

```
    printf("task6 is called!\n");
}
void task7()
{
    printf("task7 is called!\n");
}
void execute(void (*p[])(void),int n)
{
    for (int i = 0; i < n; i++) /*根据顺序调用*/
        p[i]();
}
void scheduler()
{
    /*函数指针数组储存八个任务函数*/
    void (*pa[8])(void) = { task0,task1 ,task2,task3 ,task4 ,task5 ,task6 ,task7 };
    char task[9];
    scanf("%s", task);
    int len = strlen(task);
    for (int i = 0; i < len; i++)    /*根据数字序列决定函数调用顺序*/
        p[i] = pa[task[i] - '0'];
    execute(p, len);
}
```

1.4 实验小结

本次实验学到了很多小技巧，比如双指针、指针操作字节、高精度乘法、函数指针数组的使用等，但自己对与函数指针数组的理解和运用还不太熟悉，导致在写相关的题目的时候并不是很顺利，查阅了 ppt 和相关资料之后，我对函数指针数组也有了一个较深的认识，也能比较灵活的运用。深知自己需要努力的地方还有很多，我也会继续努力的。

2 实验 7 结构与联合

2.1 表达式求值的程序验证

（1）源程序及表达式求值：

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

表达式	分析	计算值
(++p)->x	p 指针后移指向 a[1],取出成员 x	100
p++, p->c	逗号表达式取后，p 指针后移指向 a[1]，取出 c	B
*p++->t，*p->t	逗号表达式取后，p 指针后移指向 a[1]，取出 t，取内容得到 t 的第一个字符	x
*(++p)->t	p 指针后移指向 a[1]，取出 t，取内容得到 t 的第一个字符	x
*++p->t	取出 a[0]的 t，此时指针指向 U，自增后指向 V，取内容得到此时指针指向的第一个字符 V	V
++*p->t	取出 a[0]的 t，此时指针指向 U，取内容得到此时指针指向的第一个字符 U，U 自增得到 V	V

（2）验证代码：

```
#include<stdio.h>
int main()
{
    char u[7] = "UVWXYZ", v[4] = "xyz";
    struct T {
        int x;
        char c;
        char* t;
```

```
    }a[2] = { {11,'A',u},{100,'B',v} }, * p = a;
    int n;
    scanf("%d", &n);
    switch (n)
    {
    case 1:
        printf("%d", (++p)->x);
        break;
    case 2:
        printf("%c", (p++, p->c));
        break;
    case 3:
        printf("%c", (*p++->t, *p->t));
        break;
    case 4:
        printf("%c", *(++p)->t);
        break;
    case 5:
        printf("%c", *++p->t);
        break;
    case 6:
        printf("%c", ++ * p->t);
        break;
    }
    return 0;
}
```

2.2 源程序修改替换

给定一批整数，以 0 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）程序修改

```
#include<stdio.h>
#include<stdlib.h>
struct s_list {
    int data;
    struct s_list* next;
};
void create_list(struct s_list * headp, int* p);
int main() {
    struct s_list* head = NULL, * p;
    int* s = (int*)malloc(sizeof(int) * 100);
    int k = 0, i = 0;
    do {
        scanf("%d", &k);
        s[i++] = k;
    } while (k);
    create_list(head, s);
    p = head;
    while (p) {
        printf("%d\t", p->data);
        p = p->next;
    }
    printf("\n");
    free(s);
    return 0;
}
void create_list(struct s_list* headp, int* p)
{
    struct s_list* loc_head = NULL, *tail;
```

```
    if (p[0] == 0);
    else {
        loc_head=(struct s_list*)malloc(sizeof(struct s_list));
        loc_head->data = *p++;
        tail = loc_head;
        while (*p)
        {
            tail->next = (struct s_list*)malloc(sizeof(struct s_list));
            tail = tail->next;
            tail->data = *p++;
        }
        tail->next = NULL;
    }
    headp = loc_head;
}
```

错因：传入函数的只是指针的值，在函数内对指针操作并不会改变主函数中指针的值。

修改：传入二级指针。

```
void create_list(struct s_list ** headp, int* p);
create_list(&head, s);
*headp = loc_head;
```

（2）程序替换，将 create_list 函数的功能修改为创建后进先出链表

```
#include<stdio.h>
#include<stdlib.h>
struct s_list {
    int data;
    struct s_list* next;
};
void create_list(struct s_list** headp, int* p);
int main() {
```

```
    struct s_list* head = NULL, * p;
    int* s = (int*)malloc(sizeof(int) * 100);
    int k = 0, i = 0;
    do {
        scanf("%d", &k);
        s[i++] = k;
    } while (k);
    create_list(&head, s);
    p = head;
    while (p) {
        printf("%d\t", p->data);
        p = p->next;
    }
    printf("\n");
    free(s);
    return 0;
}
void create_list(struct s_list** headp, int* p)
{
    struct s_list* loc_head = NULL, *tail;
    if (p[0] == 0);
    else {
        loc_head=(struct s_list*)malloc(sizeof(struct s_list));
        loc_head->data = *p++;
        loc_head->next = NULL;
        tail = loc_head;
        while (*p)
        {
            struct s_list* q = (struct s_list*)malloc(sizeof(struct s_list));
            q->data = *p++;
```

```
            q->next=loc_head;
            loc_head=q;
        }
    }
    *headp = loc_head;
}
```

2.3 程序设计

1. 设计字段结构

算法步骤：定义一个字段结构，将一个 8 位无符号字节从最低为向最高位声明为八个字段，定义一个函数指针数组指向八个任务函数，输入一个 8 字节无符号整数，并取出二进制形式下的每一位，若该位为一，就调用对应的任务函数。

代码：

```
#include<stdio.h>
/*定义一个字段结构*/
struct bits {
    unsigned bit7 : 1;
    unsigned bit6 : 1;
    unsigned bit5 : 1;
    unsigned bit4 : 1;
    unsigned bit3 : 1;
    unsigned bit2 : 1;
    unsigned bit1 : 1;
    unsigned bit0 : 1;
}a;
/*定义八个任务函数*/
void f0(int b) {
    printf("the function %d is called!\n", b);
}
void f1(int b) {
```

```
    printf("the function %d is called!\n", b);
}
void f2(int b) {
    printf("the function %d is called!\n", b);
}
void f3(int b) {
    printf("the function %d is called!\n", b);
}
void f4(int b) {
    printf("the function %d is called!\n", b);
}
void f5(int b) {
    printf("the function %d is called!\n", b);
}
void f6(int b) {
    printf("the function %d is called!\n", b);
}
void f7(int b) {
    printf("the function %d is called!\n", b);
}
*/定义一个含有八个任务函数的函数指针数组*/
void (*p_fun[8])(int) = { f0,f1,f2,f3,f4,f5,f6,f7 };
int main()
{
    int n;
    scanf("%d", &n);
    /*取出二进制下的每一位*/
    a.bit0 = n & 1;
    a.bit1 = n >> 1 & 1;
    a.bit2 = n >> 2 & 1;
```

```
    a.bit3 = n >> 3 & 1;
    a.bit4 = n >> 4 & 1;
    a.bit5 = n >> 5 & 1;
    a.bit6 = n >> 6 & 1;
    a.bit7 = n >> 7 & 1;
    /*如果某一位是 1 就调用对应函数*/
    if (a.bit0 == 1) p_fun[0](0);
    if (a.bit1 == 1) p_fun[1](1);
    if (a.bit2 == 1) p_fun[2](2);
    if (a.bit3 == 1) p_fun[3](3);
    if (a.bit4 == 1) p_fun[4](4);
    if (a.bit5 == 1) p_fun[5](5);
    if (a.bit6 == 1) p_fun[6](6);
    if (a.bit7 == 1) p_fun[7](7);
    return 0;
}
```

2. 班级成绩单

算法步骤：主函数中用 if-else 实现功能选择，对每个功能函数：输入信息函数，每次调用只创建一个节点加在链表最后，如果要输入多个学生的信息就在主函数中利用循环调用多次；对输出信息函数，遍历链表，打印出每一个结点的信息；对修改信息函数，遍历链表，找出要修改的学生，然后直接修改即可；对统计平均成绩函数，遍历链表，算出每一个学生的平均成绩，然后输出即可；对输出所有信息函数，遍历链表，打印出每个学生的所有信息即可。

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/*学生信息结点*/
typedef struct student
{
```

```
    struct _stu
    {
        char num[11];
        char name[11];
        int score[4];
    }stu;
    struct student* next;
}stnode;
void input(stnode** a)
{
    stnode* p = *a;
    while (p->next) p = p->next;
    stnode* sim = (stnode*)malloc(sizeof(stnode));
    p->next = sim;
    scanf("%s%s", sim->stu.num, sim->stu.name);
    for (int i = 0; i < 4; i++)
        scanf("%d", &sim->stu.score[i]);
    sim->next = NULL;
}
void print(stnode* a)
{
    stnode* p = a->next;
    while (p)
    {
        printf("%s %s %d %d %d %d\n", p->stu.num, p->stu.name, p->stu.score[0],
            p->stu.score[1], p->stu.score[2], p->stu.score[3]);
        p = p->next;
    }
}
void modify(stnode** a, char *s,int i, int e)
```

```
{
    stnode* p = (*a)->next;
    while (p && strcmp(s, p->stu.num))
        p = p->next;
    p->stu.score[i - 1] = e;
}
void ave(stnode* a)
{
    stnode* p = a->next;
    while (p)
    {
        double ave = 0;
        for (int i = 0; i < 4; i++)
            ave += p->stu.score[i];
        ave /= 4;
        printf("%s %s %.2lf\n", p->stu.num, p->stu.name, ave);
        p = p->next;
    }
}
void all(stnode* a)
{
    stnode* p = a->next;
    while (p)
    {
        double ave=0;
        int sum = 0;
        for (int i = 0; i < 4; i++)
            ave += p->stu.score[i];
        sum = ave;
        ave /= 4;
```

```
        printf("%s %s %d %.2lf\n", p->stu.num, p->stu.name, sum,ave);
        p = p->next;
    }
}
int main()
{
    stnode* a=(stnode *)malloc(sizeof(stnode));
    a->next = NULL;
    int op=1,num,i,loc,score;
    char temp[100];
    scanf("%d", &op);
    while (1)
    {
        /*功能选择*/
        if(op==0)
            break;
        else if(op==1)
        {
            scanf("%d", &num);
            /*一次只输入一个学生成绩，方便后续的添加*/
            for (i = 0; i < num; i++)
                input(&a);
        }
        else if(op==2)
        {
            print(a);
        }
        else if(op==3)
        {
            scanf("%s%d%d", temp, &loc, &score);
```

```
            modify(&a, temp, loc, score);
        }
        else if(op==4)
        {
            ave(a);
        }
        else if(op==5)
        {
            all(a);
        }
        scanf("%d", &op);
    }
    return 0;
}
```

3. 成绩排序（一）

算法步骤：修改学生信息结点结构和上题函数，在输入学生信息的时候就把平均成绩和总成绩算出放入结点数值域，在调用每个输出函数的时候首先根据平均成绩进行冒泡排序（冒泡排序较普遍，此处不做赘述），然后进行输出，在调用修改信息函数的时候，先修改信息，再重新排序。

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/*修改了学生信息结构，增加了总分和平均分*/
struct _stu
{
    char num[11];
    char name[11];
    int score[4];
    int sum;
```

```
    double ave;
};
typedef struct student
{
    struct _stu stu;
    struct student* next;
}stnode;
void mysort(stnode**a) /*排序函数，本质上是冒泡排序*/
{
      stnode* p1, * p2;
      for (p1 = (*a)->next; p1; p1 = p1->next)
          for (p2 = p1->next; p2; p2 = p2->next)
          {
              struct _stu temp;
              if (p1->stu.ave > p2->stu.ave)
              {
                  temp = p1->stu;
                  p1->stu = p2->stu;
                  p2->stu = temp;
              }
          }
}
void input(stnode** a)
{
    stnode* p = *a;
    while (p->next) p = p->next;
    stnode* sim = (stnode*)malloc(sizeof(stnode));
    p->next = sim;
    scanf("%s%s", sim->stu.num, sim->stu.name);
    sim->stu.ave = 0;
```

```
        sim->stu.sum = 0;
        for (int i = 0; i < 4; i++)
        {
            scanf("%d", &sim->stu.score[i]);
            sim->stu.ave +=sim->stu.score[i];
        }
        sim->stu.sum = sim->stu.ave;
        sim->stu.ave /= 4;
        sim->next = NULL;
    }
    void print(stnode** a)
    {
        mysort(a); /*先排序后输出*/
        stnode *p = (*a)->next;
        while (p)
        {
            printf("%s %s %d %d %d %d\n", p->stu.num, p->stu.name,
p->stu.score[0], p->stu.score[1], p->stu.score[2], p->stu.score[3]);
            p = p->next;
        }
    }
    void modify(stnode** a, char* s, int i, int e)
    {
        stnode* p = (*a)->next;
        while (p && strcmp(s, p->stu.num))
            p = p->next;
        p->stu.sum = p->stu.sum - p->stu.score[i - 1] + e;
        p->stu.ave = p->stu.sum;
        p->stu.ave /= 4;
        p->stu.score[i - 1] = e;
```

```
        mysort(a); /*修改后重新排序*/
    }
    void ave(stnode** a)
    {
        stnode* p;
        mysort(a); /*先排序后输出*/
        p = (*a)->next;
        while (p)
        {
            printf("%s %s %.2lf\n", p->stu.num, p->stu.name, p->stu.ave);
            p = p->next;
        }
    }
    void all(stnode** a)
    {
        stnode* p;
        mysort(a); /*先排序后输出*/
        p = (*a)->next;
        while (p)
        {
            printf("%s %s %d %.2lf\n", p->stu.num, p->stu.name, p->stu.sum,
p->stu.ave);
            p = p->next;
        }
    }
    int main()
    {
        stnode* a = (stnode*)malloc(sizeof(stnode));
        a->next = NULL;
        int op = 1, num, i, loc, score;
```

```
char temp[100];
scanf("%d", &op);
while (1) /*功能选择*/
{
    if (op == 0)
        break;
    else if (op == 1)
    {
        scanf("%d", &num);
        for (i = 0; i < num; i++)
            input(&a); /*逐个输入*/
    }
    else if (op == 2)
    {
        print(&a);

    }
    else if (op == 3)
    {
        scanf("%s%d%d", temp, &loc, &score);
        modify(&a, temp, loc, score);
    }
    else if (op == 4)
    {
        ave(&a);
    }
    else if (op == 5)
    {
        all(&a);
    }
```

```
        scanf("%d", &op);
    }
    return 0;
}
```

4. 回文字符串：

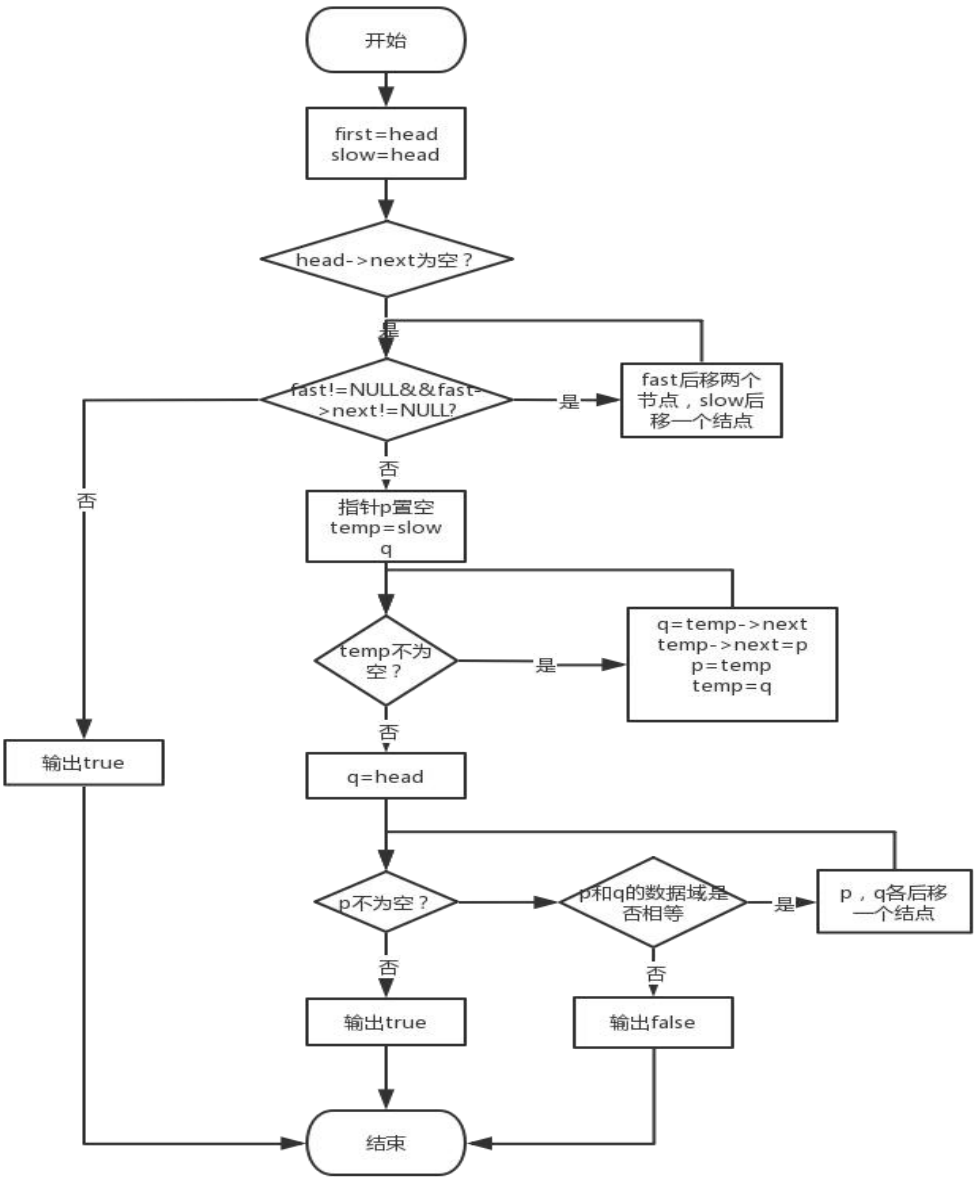

图 2-1 程序设计题 4 流程图

算法步骤：首先创建链表储存字符串的每个字符。然后进入判断函数，设置快慢指针，并移动指针，快指针一次移动两个结点，慢指针一次移动一个结点，

当快指针指向 NULL（偶数个字符）或快指针的 next 指向 NULL（奇数个字符）时，慢指针指向字符串后半段的第一个字符（偶数个字符）或中间字符（奇数个字符），然后通过将后面的结点插入到前面结点的前方，实现后半段链表翻转。然后从前半段第一个字符和后半段第一个字符开始依次比较，不相同就输出 false，如果遍历完成，说明符合回文，输出 true。

代码：

```
void createLinkList(C_NODE** headp, char s[])
{
    *headp = (C_NODE*)malloc(sizeof(C_NODE));
    C_NODE* p = (*headp);
    int i = 0;
    p->data = s[i];
    i++;
    while (s[i])
    {
        C_NODE* q = (C_NODE*)malloc(sizeof(C_NODE));
        p->next = q;
        q->next = NULL;
        q->data = s[i];
        p = q;
        i++;
    }
}
void judgePalindrome(C_NODE* head)
{
    C_NODE* fast = head, * slow = head;
    if (head->next == NULL)
    {
        printf("true"); /*说明是空串，一定回文*/
        return;
```

```
}
else
{
    while (fast && fast->next) /*fast 为空对应偶数个字符，fast->next 为
        空对应奇数个字符*/
    {
        fast = fast->next->next;
        slow = slow->next;
    }
    C_NODE* p = NULL, * temp = slow, * q;
    /*将 slow 指向的结点及之后的结点组成的子链表翻转，此时 slow
    指向后半段第一个字符（偶数个字符）或者中间字符（奇数个字符，
    当为奇数时，翻转之后前半段的最后一个字符与中间字符之间的连
    接并没有断开，所以前半段和后半段均可以遍历到中间字符，避免
    错误的发生*/
    while (temp)
    {
       /*实质是将后面的结点插到前面的结点的前方，实现反转*/
        q = temp->next;
        temp->next = p;
        p = temp;
        temp = q;
    }/*循环退出后 p 指向后半段第一个字符*/
    q = head; /*让 q 指向前半段第一个字符，用 q，p 遍历两段链表*/
    while (p) /*p 为空时完成遍历*/
    {
        if (p->data != q->data) /*不相等就输出 false 并退出*/
        {
            printf("false");
            return;
```

```
            }
            /*p，q 后移，判断后一个字符*/
            p = p->next;
            q = q->next;
        }
       /*循环可以执行完成，说明符合回文*/
        printf("true");
        return;
    }
}
```

5. 成绩排序（二）

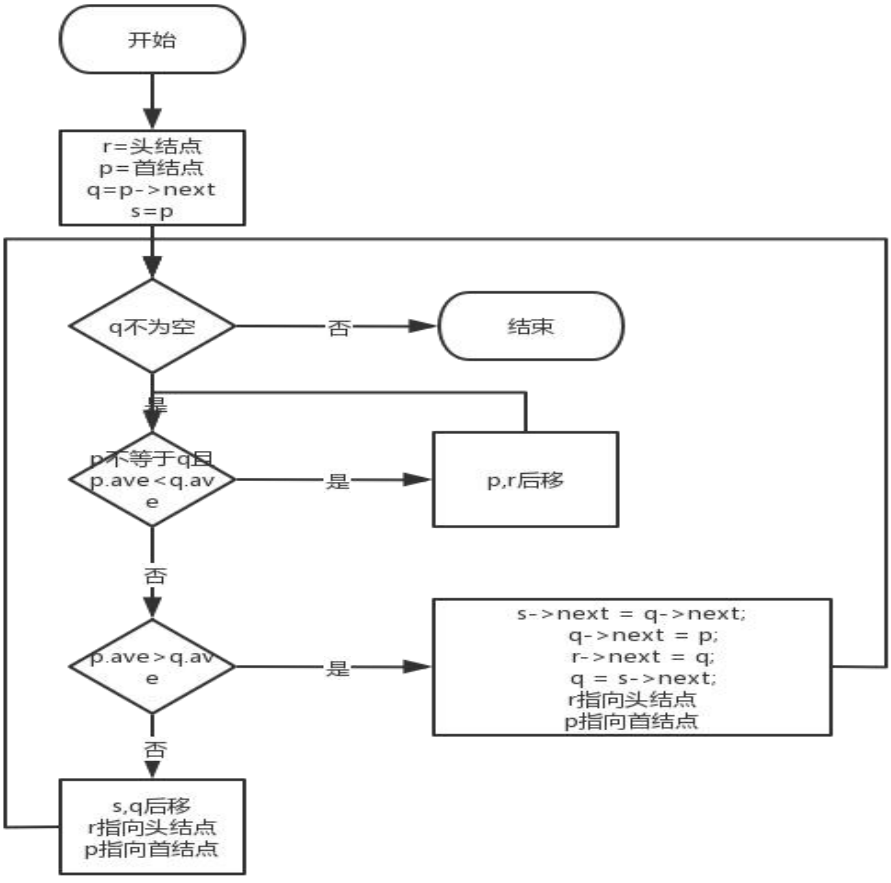

图 2-2 程序设计题 5 流程图

算法步骤：这里只叙述排序思想。定义四个指针，初始化时 r 指向头结点，

p 指向首结点，s 等于 p，q 等于 s->next。让 s，q 遍历链表，当 q 指向某个结点时，r，p 开始遍历，当 p 指向的结点的数据大于或等于 q 的数据时用 s->next 预存 q->next 将 q 插入 r，p 之间，然后让 q 指向 s->next，通过依次比较并实现插入排序。

代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct _stu
{
    char num[11];
    char name[11];
    int score[4];
    int sum;
    double ave;
};
typedef struct student
{
    struct _stu stu;
    struct student* next;
}stnode;
/*排序函数*/
void mysort(stnode **a)
{
    stnode* p = (*a)->next, * q = p->next, *s=p, * r = (*a);
    /*相关指针初始化*/
    while (q) /*q 是用来遍历链表的，q 为空说明已经排序完成*/
    {
        while (p != q && p->stu.ave < q->stu.ave)
      /*对于 q 指向的每个结点，用 p 从头开始遍历并找到第一个大于等
```

```
            于 q.ave 的结点*/
            {
                p = p->next;
                r = r->next;
            }

            if (p->stu.ave > q->stu.ave)
            /*通过指针将 q 结点插到 r，p 结点之间*/
            {
                s ->next= q->next;
                q->next = p;
                r->next = q;
                q = s->next;
                r = (*a);
                p = (*a)->next;
            }
            else   /*二者相等，不用插入，s，q 后移，r，p 归位即可*/
            {
                s = s->next;
                q = q->next;
                r = (*a);
                p = (*a)->next;
            }
        }
    }
void input(stnode * *a)
{
    stnode* p = *a;
    while (p->next) p = p->next;
    stnode* sim = (stnode*)malloc(sizeof(stnode));
```

```
        p->next = sim;
        scanf("%s%s", sim->stu.num, sim->stu.name);
        sim->stu.ave = 0;
        sim->stu.sum = 0;
        for (int i = 0; i < 4; i++)
        {
            scanf("%d", &sim->stu.score[i]);
            sim->stu.ave += sim->stu.score[i];
        }
        sim->stu.sum = sim->stu.ave;
        sim->stu.ave /= 4;
        sim->next = NULL;
    }
    void print(stnode **a)
    {
        mysort(a);
        stnode *p = (*a)->next;
        while (p)
        {
            printf("%s %s %d %d %d %d\n", p->stu.num, p->stu.name,
p->stu.score[0], p->stu.score[1], p->stu.score[2], p->stu.score[3]);
            p = p->next;
        }
    }
    void modify(stnode * *a, char* str, int i, int e)
    {
        stnode* p = (*a)->next;
        while (p && strcmp(str, p->stu.num))
            p = p->next;
        p->stu.sum = p->stu.sum - p->stu.score[i - 1] + e;
```

```
            p->stu.ave = p->stu.sum;
            p->stu.ave /= 4;
            p->stu.score[i - 1] = e;
            mysort(a);
        }
        void ave(stnode * *a)
        {
            mysort(a);
            stnode *p = (*a)->next;
            while (p)
            {
                printf("%s %s %.2lf\n", p->stu.num, p->stu.name, p->stu.ave);
                p = p->next;
            }
        }
        void all(stnode * *a)
        {
            mysort(a);
            stnode *p = (*a)->next;
            while (p)
            {
                printf("%s %s %d %.2lf\n", p->stu.num, p->stu.name, p->stu.sum,
p->stu.ave);
                p = p->next;
            }
        }
        int main()
        {
            stnode* a = (stnode*)malloc(sizeof(stnode));
            a->next = NULL;
```

```
int op = 1, num, i, loc, score;
char temp[100];
scanf("%d", &op);
while (1)
{
    if (op == 0)
        break;
    else if (op == 1)
    {
        scanf("%d", &num);
        for (i = 0; i < num; i++)
            input(&a);
    }
    else if (op == 2)
    {
        print(&a);

    }
    else if (op == 3)
    {
        scanf("%s%d%d", temp, &loc, &score);
        modify(&a, temp, loc, score);
    }
    else if (op == 4)
    {
        ave(&a);
    }
    else if (op == 5)
    {
        all(&a);
```

```
            }
            scanf("%d", &op);
        }
        return 0;
    }
```

6. 逆波兰表达式求值

算法步骤：设计栈数据结构用于储存数值。遍历输入的逆波兰表达式，每次遇到数字时，将空格前的数值算出并入栈（负数需要特判），当遇到操作符时，让栈中最上面的两个数出栈并进行运算，将结果入栈，直到表达式遍历完成，栈顶元素就是最后结果。

代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
/*栈结构用来储存数字*/
typedef struct sqstack {
   int* num;
   int top;
} Sqstack;
void initail(Sqstack** s) {              /*初始化栈*/
   (*s)->num = (int*)malloc(sizeof(int) * 10);
   (*s)->top = -1;
}
void push(Sqstack** s, int e) {       /*入栈*/
   (*s)->num[++((*s)->top)] = e;
}
void pop(Sqstack** s, int* e) {       /*出栈*/
   *e = (*s)->num[((*s)->top)--];
```

```
}
int main() {
    char str[100];
    int num[10] = { 0 };
    int k = 0,m=0;
    fgets(str, 100, stdin); /*输入逆波兰表达式*/
    Sqstack* s=(Sqstack *)malloc(sizeof(Sqstack));
    initail(&s);
    int i = 0;
    while (str[i]) {
        if (str[i] >= '0' && str[i] <= '9') {
            /*当前字符为数字时，继续往后遍历*/
            while (str[i] >= '0' && str[i] <= '9') {
                num[k++] = str[i++] - '0';    /*储存数字*/
                if (str[i] == ' ') {    /*当前数字为空格时，算出前面的数字*/
                    int temp = 0;
                    for (int j = k - 1; j >= 0; j--)
                        temp += num[j] * pow(10, k - j - 1);
                    push(&s, temp);    /*数字入栈*/
                    memset(num, 0, sizeof(num));    /*数字数组归零*/
                    k = 0;
                    i++;
                    break;
                }
            }
        }
        else if (str[i] == '-' && str[i + 1] >= '0' && str[i + 1] <= '9') {
            i++;    /*当前字符为'-'且后一个字符是数字，说明是负数，后同前*/
            while (str[i] >= '0' && str[i] <= '9') {
                num[k++] = str[i++] - '0';
```

```
            if (str[i] == ' ') {
                int temp = 0;
                for (int j = k - 1; j >= 0; j--)
                    temp += num[j] * pow(10, k - j - 1);
                push(&s, -temp);
                memset(num, 0, sizeof(num));
                k = 0;
                i++;
                break;
            }
        }
    }
    else if (str[i] == ' ') {     /*若为空格，直接遍历下一字符*/
        i++;
        continue;
    }
    else if (str[i] == '+' || str[i] == '-' || str[i] == '*' || str[i] == '/') {
        int n1, n2;
        switch (str[i]) {     /*当前字符为操作符是，栈顶的两个元素出栈并运
                                算，结果入栈*/
        case '+':
            pop(&s, &n2);
            pop(&s, &n1);
            push(&s, n1 + n2);
            break;
        case '-':
            pop(&s, &n2);
            pop(&s, &n1);
            push(&s, n1 - n2);
            break;
```

```
            case '*':
                pop(&s, &n2);
                pop(&s, &n1);
                push(&s, n1 * n2);
                break;
            case '/':
                pop(&s, &n2);
                pop(&s, &n1);
                push(&s, n1 / n2);
                break;
            }
            i++;
        }
        else break;
    }
    int result;
    pop(&s, &result);   /*遍历结束栈顶元素即为最后结果，使其出栈并输出*/
    printf("%d", result);
    return 0;
}
```

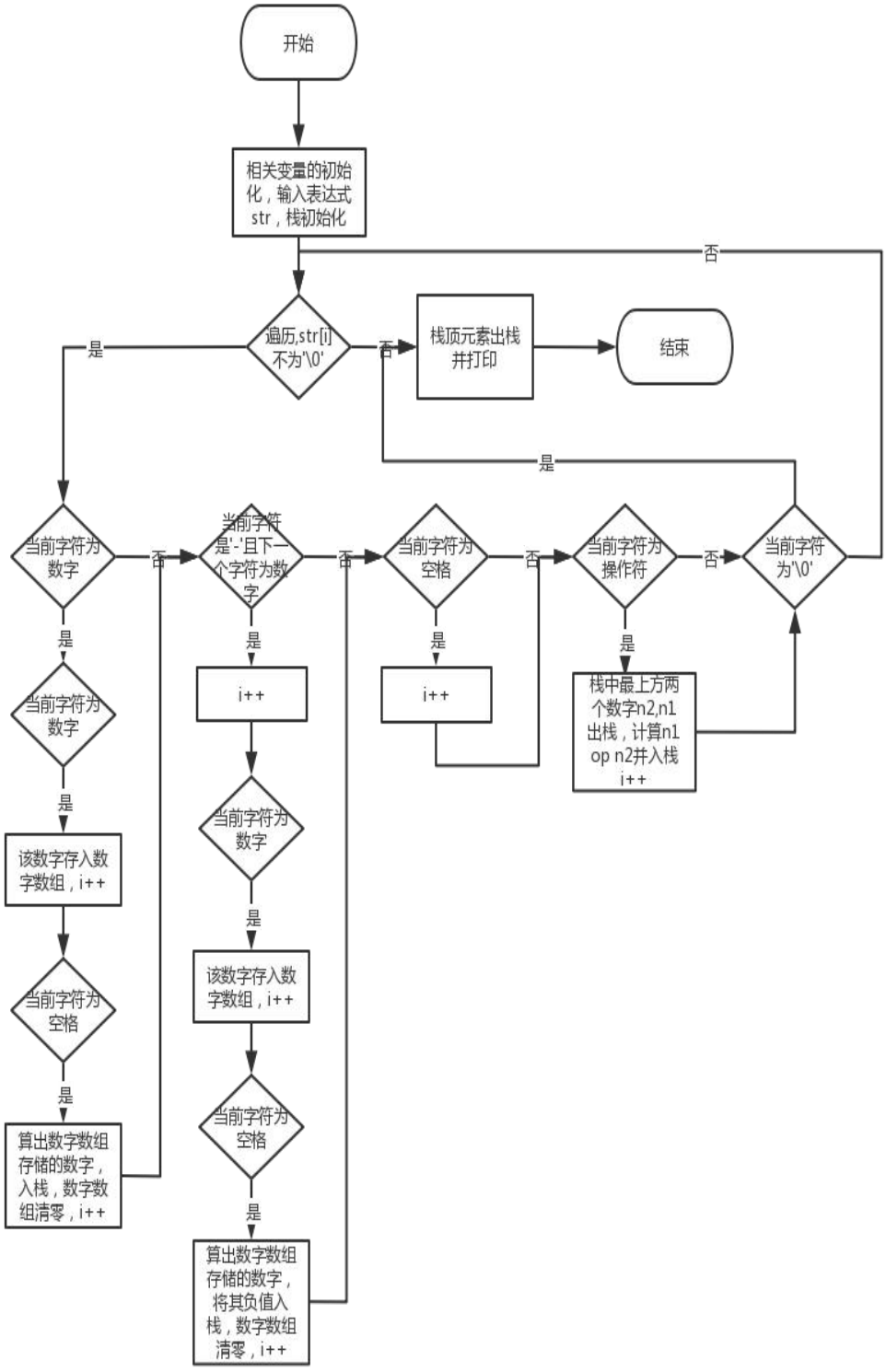

图 2-3 程序设计题 6 流程图

2.4 实验小结

个人感觉本次实验还是有一一定的难度。

程序设计题的设计字段题目让我加强了对位运算的掌握。

班级成绩单系列题目中的成绩排序，由于自己没有考虑到调用功能的任意性，使输出结果并没有保证全部排好序，然后把排序加到每一次输出之前和每一次修改之后，就能顺利得到正确结果。

这次第一次写了交换指针域来进行链表排序，个人感觉利用冒泡排序有些繁琐，故选择了插入排序，让自己对指针和链表的操作更加熟练了一些。

利用链表来判断回文字符串对我来说也是一次挑战。经过自己思考并参阅资料之后，选择了快慢指针+反转链表的方式来实现回文的判断。

逆波兰表达式求值的解题思路比较清晰，但起初并没有考虑到负数的问题，在同学的提醒之下成功通过了本题。

参考文献

[1] 曹计昌,卢萍,李开. C 语言与程序设计,北京：电子工业出版社,2013

[2] 卢萍,李开,王多强，甘早斌. C 语言程序设计典型题解与实验指导,北京：清华大学出版社,2019