

华中科技大学

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 2505

学 号 U202514843

姓 名 徐靖杰

指导教师 郝义学

报告日期 2026 年 1 月 3 日

计算机科学与技术学院

目 录

1 实验 6 指针程序设计实验.....	1
1.1 程序改错与跟踪调试.....	1
1.2 程序完善与修改替换第 (1) 题.....	4
1.3 程序完善与修改替换第 (2) 题.....	6
1.4 程序设计 1: 通过指针取出字节	8
1.5 程序设计 2: 删 除重 复元素	10
1.6 程序设计 3: 矩阵逆时针旋转 90°	12
1.7 程序设计 4: 命令行实现对 N 个整数排序	14
1.8 程序设计 5: 删 除子串	17
1.9 程序设计 6: 非负整数积	19
1.10 程序设计 7: 函数调度	22
1.11 小结.....	24
2 实验 7 结构与联合	26
2.1 表达式求值的程序验证	26
2.2 源程序修改替换 (一)	29
2.3 源程序修改替换 (二)	34
2.4 程序设计 1: 设计字段结构.....	37
2.5 程序设计 2: 班级成绩单	39
2.6 程序设计 3: 成绩排序 (一)	45
2.7 程序设计 4: 成绩排序 (二)	48
2.8 程序设计 5: 回文字符串判断	50
2.9 程序设计 6: 逆波兰表达式.....	53
2.10 小结.....	60

1 实验 6 指针程序设计实验

1.1 程序改错与跟踪调试

1.1.1 原代码

```
1 #include<stdio.h>
2 char *strcpy(char *, const char *);
3 int main(void) {
4     char *s1, *s2, *s3;
5     printf("Input a string:\n");
6     scanf("%s", s2);
7     strcpy(s1, s2);
8     printf("%s\n", s1);
9     printf("Input a string again:\n");
10    scanf("%s", s2);
11    s3 = strcpy(s1, s2);
12    printf("%s\n", s3);
13    return 0;
14 }
15 /* 字符串复制函数 */
16 char *strcpy(char *t, const char *s) {
17     while ((*t++ = *s++));
18     return (t);
19 }
```

Listing 1 待改错的程序

1.1.2 程序改错与跟踪调试

1. 错误 1 (第 4 行): 指针变量未初始化

- 错误原因: s1, s2, s3 声明为字符指针, 但没有为它们分配内存空间。这些指针指向随机的内存地址, 直接使用会导致未定义行为 (段错误等)。

```
Input a string:  
programming  
-----  
Process exited after 40.7 seconds with return value 3221225477  
请按任意键继续... |
```

图 1-1 段错误

- 可能后果：如图 1-1，返回值 3221225477 表明发生了段错误。

2. 错误 2（第 18 行）：strcpy 函数返回错误地址

- 错误原因：在 while 循环结束后，t 指针已经移动到字符串结束符 '\0' 之后的位置。返回 t 实际上返回的是目标字符串的结束位置，而不是起始位置。
- 跟踪调试：此处使用 DEV C++ 编译器，在第 17 行设置断点，查看 t 的值。如图 1-2、1-3 和 1-4，每一次循环时 t 指针都在移动。



图 1-2 第一次执行



图 1-3 第二次执行



图 1-4 第三次执行

- 可能后果 main 函数中的 s3 将指向错误的内存地址，后续的 printf("%s\n", s3) 可能会输出乱码或导致程序崩溃。

1.1.3 改正方案

```
1 #include<stdio.h>
2 char *strcpy(char *,const char *);
3 int main(){
```

华中科技大学课程实验报告

```
4     char s1[100];
5     char s2[100];
6     char *s3;
7     scanf("%s",s2);
8     strcpy(s1,s2);
9     printf("%s\n",s1);
10    scanf("%s",s2);
11    s3=strcpy(s1,s2);
12    printf("%s\n",s3);
13    return 0;
14 }
15 char *strcpy(char *t,const char *s){
16     char *start = t;
17     while ((*t++ = *s++));
18     return start;
19 }
```

Listing 2 改错后的程序

改正后的程序主要解决了前文所指出的两个关键问题：1. **内存分配问题**：- 将未初始化的字符指针 s1, s2, s3 改为字符数组 s1[100] 和 s2[100]，确保有足够的空间存储输入字符串。- s3 仍然保留为指针，但它指向的是 s1 数组的有效地址。

2. **函数返回值问题**：- 在 strcpy 函数开头保存目标字符串的起始地址 start = t。- 循环结束后返回 start，确保调用者获得正确的字符串起始位置。
- 保持了 while ((*t++ = *s++)) 的简洁写法，这是 C 语言中字符串拷贝的经典实现。

1.2 程序完善与修改替换第（1）题

1.2.1 程序完善填空

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4 /* 对指针数组s指向的size个字符串进行升序排序 */
5 void strsort ( char *s[ ],int size )
6 {
7     char * temp;
8     int i, j ;
9     for(i=0; i<size-1; i++)
10    {
11        for(j=0; j<size-i-1; j++) {
12            if (strcmp(s[j],s[j+1])>0)
13            {
14                temp=s[j];
15                s[j]=s[j+1];
16                s[j+1]=temp;
17            }
18        }
19    }
20
21 int main( )
22 {
23     int N;
24     scanf ("%d",&N);
25     N+=1;
26     int i;
27     char *s[N], t[50];
28     for(i=0;i<N;i++)
29     {
30         gets(t);
31         s[i] = (char *)malloc(strlen(t)+1);
32         strcpy(s[i],t);
33     }
34     strsort (s,N);
35     for(i=0;i<N;i++) puts(s[i]);
36     return 0;
37 }
```

Listing 3 实验 6 程序完善与修改替换第（1）题完善后的程序

分析思路

1. stdlib.h: 程序中使用了 malloc 函数进行动态内存分配。
2. char *: 用于交换字符串指针的临时变量，类型必须匹配。
3. strcmp(s[j], s[j+1])>0: 比较相邻字符串大小，若前者大于后者则交换。
4. s[j]=s[j+1]: 交换指针步骤之一，将 s[j] 指向 s[j+1] 的内容。
5. s[i], t: 将输入的字符串 t 复制到动态分配的内存中。
6. s, N: 调用排序函数，传入指针数组和字符串个数两个参数。

```
3
C
Python
Java

C
Java
Python

-----
Process exited after 7.012 seconds with return value 0
请按任意键继续. . .
```

图 1-5 运行截图

1.2.2 修改替换

题目要求请用二级指针形参重写第 2 关的 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。此处使用的具体方案如下：

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

Listing 4 重写后的 strsort 函数

1.3 程序完善与修改替换第（2）题

1.3.1 程序完善填空

```
1 #include<stdio.h>
2 #include<string.h>
3 #include<stdlib.h>
4 int main()
5 {
6     char*(*p)(char a[],char b[]);
7     char a [80],b[80],*result;
8     int choice;
9     while(1){
10         do {
11             scanf("%d",&choice);
12         }while(choice<1 || choice>4);
13         switch(choice)
14         {
15             case 1 :p =strcpy; break;
16             case 2 :p =strcat; break;
17             case 3 :p =strtok; break;
18             case 4 :goto down;
19         }
20         getchar();
21         gets(a);
22         gets(b);
23         result = p(a,b);
24         printf("%s\n",result);
25     }
26     down :return 0;
27 }
```

Listing 5 实验 6 程序完善与修改替换第（2）题完善后的程序

1.3.2 分析思路

1. `char*(*p)(char a[],char b[])`: 定义函数指针，用于指向不同的字符串处理函数。
2. `gets(a)`: 读取第一个字符串输入。
3. `gets(b)`: 读取第二个字符串输入。

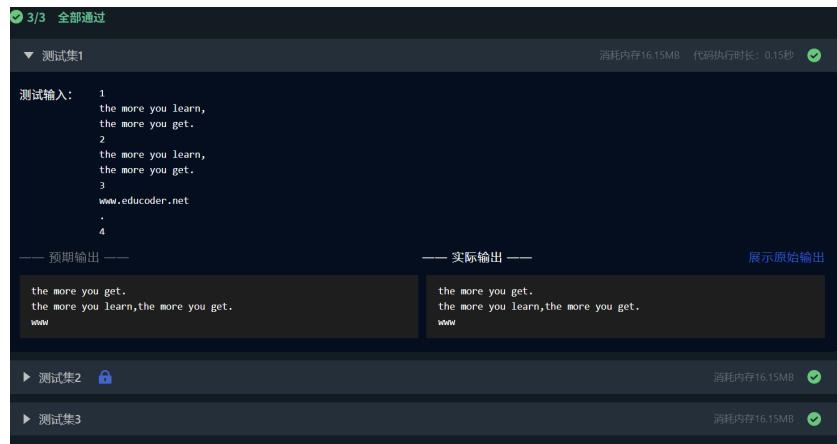


图 1-6 运行截图

4. p: 通过函数指针调用选中的字符串处理函数。

1.3.3 修改替换

题目要求使用转移表而不是 switch 语句重写第 4 关程序。此处使用的具体方案如下。

```
1 #include<stdio.h>
2 #include<string.h>
3 int main(){
4     int temp;
5     char s1[100];    char s2[100];    char *s3;
6     char * (*funlist[3])(char *,const char *)={
7         strcpy, strcat, strtok
8     };
9     scanf("%d",&temp);
10    getchar();
11    while(temp!=4){
12        fgets(s1,sizeof(s1),stdin);
13        fgets(s2,sizeof(s2),stdin);
14        s1[strcspn(s1, "\n")] = '\0';
15        s2[strcspn(s2, "\n")] = '\0';
16        s3=funlist[temp-1](s1,s2);
17        printf("%s\n",s3);
18        scanf("%d",&temp);
19        getchar();
20    }
21    return 0;
22 }
```

Listing 6 实验 6 程序完善与修改替换第 (2) 题完善后的程序

1.4 程序设计 1：通过指针取出字节

1.4.1 题目要求

编写一个程序，从整形变量的高字节开始，依次取出每字节的高 4 位和低 4 位并以十六进制数字字符的形式进行显示，要求通过指针取出每字节。

1.4.2 设计程序流程图

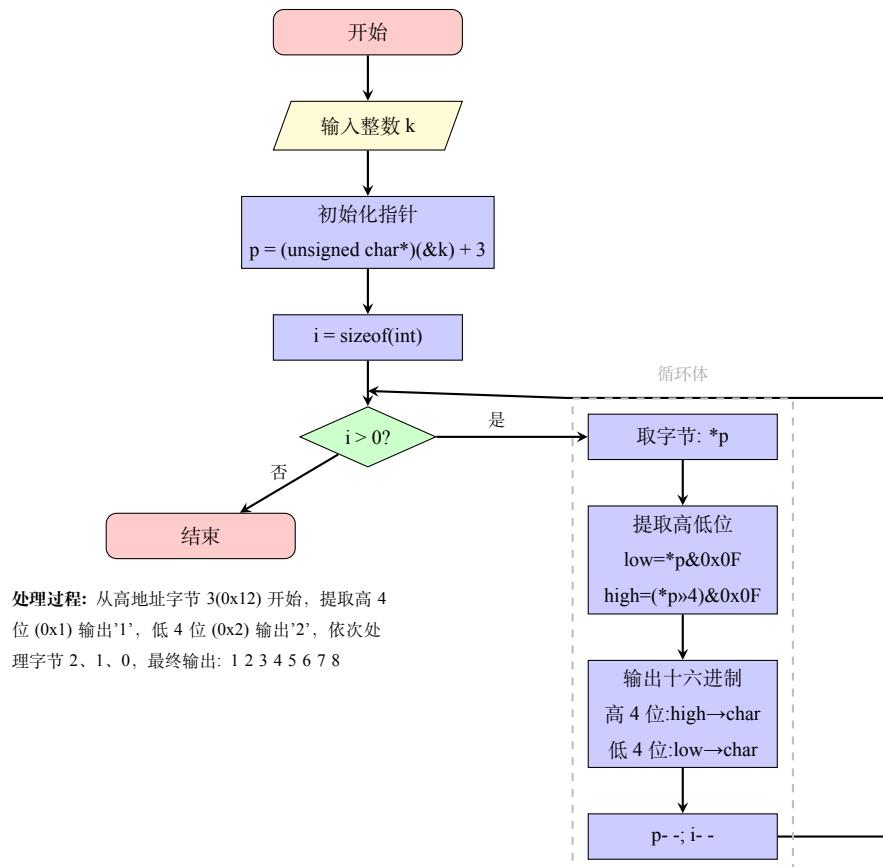


图 1-7 整数按字节分解为十六进制字符的流程图

1.4.3 完整程序代码

```

1 #include<stdio.h>
2 int main(){
3     int k;
4     scanf("%d", &k);
5     unsigned char * p=(unsigned char*)(&k)+3;
6     for(int i[sizeof(int)]; i>0; i--){
7         unsigned char low = *p & 0x0F;

```

```
8     unsigned char high = (*p >> 4) & 0x0F;
9     printf("%c ", (high <= 9) ? high + '0' : high - 10 + 'a');
10    printf("%c ", (low <= 9) ? low + '0' : low - 10 + 'a');
11    p--;
12 }
13 return 0;
14 }
```

Listing 7 实验 6 程序设计 1 完整代码

1.4.4 程序运行示例

```
123456
0 0 0 1 e 2 4 0
-----
Process exited after 2.771 seconds with return value 0
请按任意键继续. . . |
```

图 1-8 通过指针取出字节

1.5 程序设计 2：删除重复元素

1.5.1 题目要求

去掉有 n 个元素的有序整数序列 a 中的重复元素，返回去重后序列的长度。

1.5.2 设计程序流程图

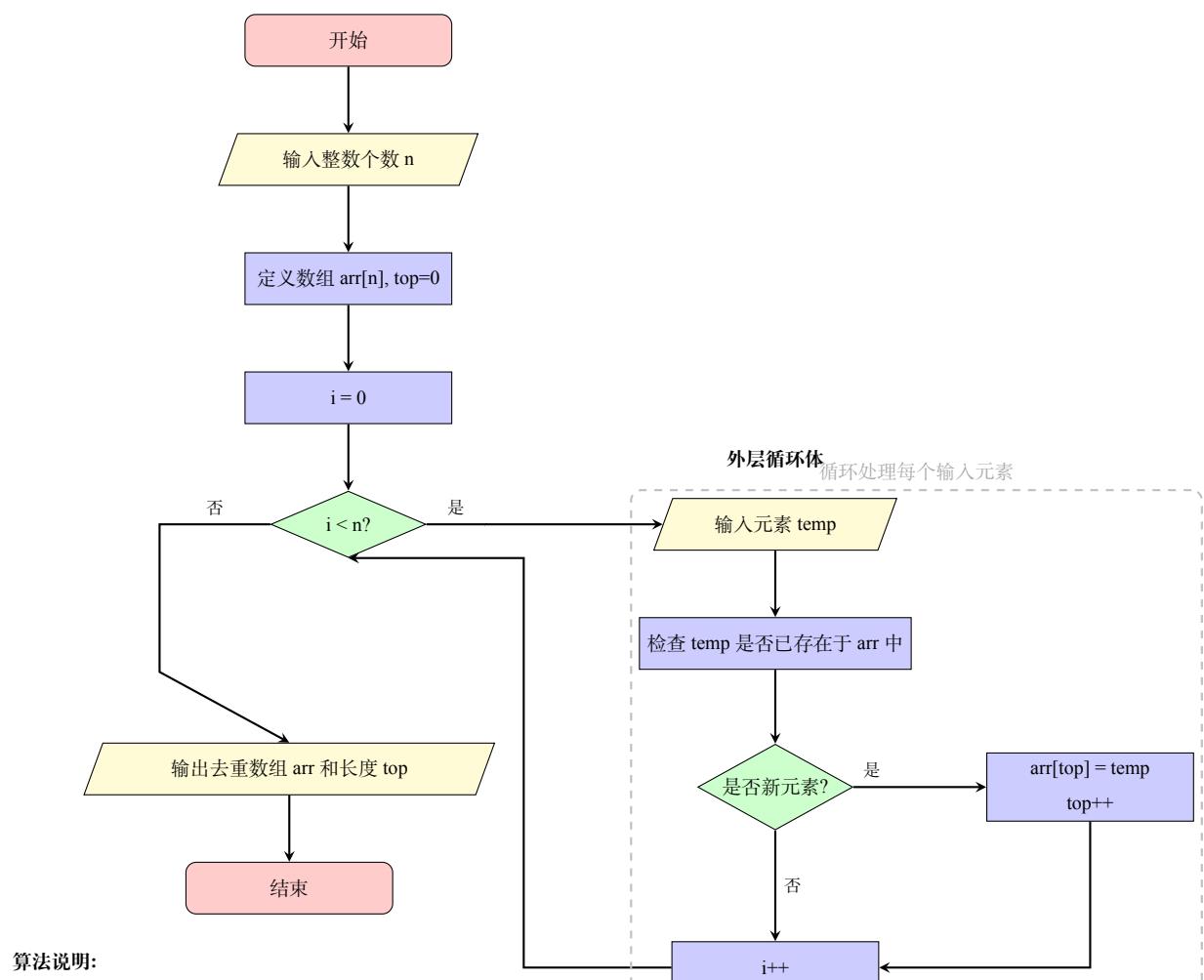


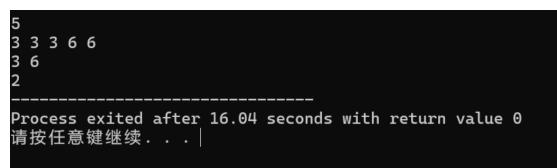
图 1-9 有序整数序列去重算法流程图

1.5.3 完整程序代码

```
1 #include<stdio.h>
2 int main(){
3     int n;
4     scanf("%d",&n);
5     int arr[n];
6     int top=0;
7     int temp,flag=0;
8     for(int i=0;i<n;i++){
9         flag=0;
10        scanf("%d",&temp);
11        for(int j=0;j<top;j++){
12            if(temp==*(arr+j)){
13                flag=1;
14                break;
15            }
16        }
17        if(!flag){
18            arr[top]=temp;
19            top+=1;
20        }
21    }
22    for(int i=0;i<top-1;i++){
23        printf("%d ",*(arr+i));
24    }
25    printf("%d",*(arr+top-1));
26    printf("\n%d",top);
27    return 0;
28 }
```

Listing 8 实验 6 程序设计 1 完整代码

1.5.4 程序运行示例



```
5
3 3 3 6 6
3 6
2
-----
Process exited after 16.04 seconds with return value 0
请按任意键继续. . . |
```

图 1-10 删除重复元素

1.6 程序设计 3：矩阵逆时针旋转 90°

1.6.1 题目要求

输入图像矩阵的行数 n 和列数 m，然后输入 n 行 m 列的整数矩阵，输出原始矩阵逆时针旋转 90° 后的矩阵。

1.6.2 设计程序流程图

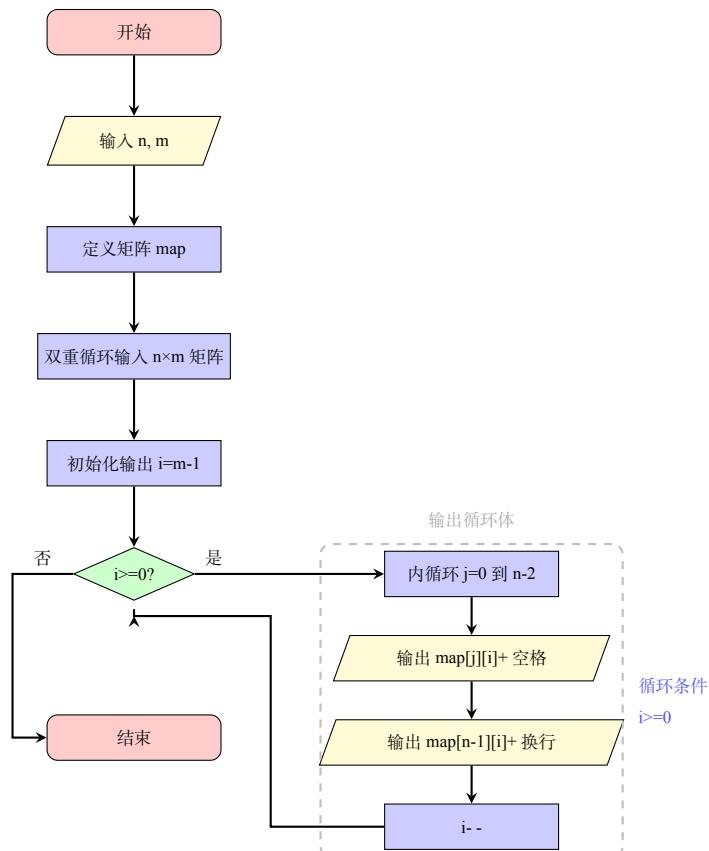


图 1-11 矩阵逆时针旋转 90° 算法流程图（简化版）

1.6.3 完整程序代码

```
1 #include<stdio.h>
2 int main(){
3     int n,m;
4     scanf("%d%d",&n,&m);
5     int map[n][m];
6     for(int i=0;i<n;i++){
7         for(int j=0;j<m;j++){
8             scanf("%d",&map[i][j]);
9         }
10    }
11    for(int i=m-1;i>=0;i--){
12        for(int j=0;j<n-1;j++){
13            printf("%d ",map[j][i]);
14        }
15        printf("%d\n",map[n-1][i]);
16    }
17    return 0;
18 }
```

Listing 9 矩阵逆时针旋转 90° 完整代码

1.6.4 程序运行示例

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

-----
Process exited after 6.117 seconds with return value 0
请按任意键继续. . . |
```

图 1-12 矩阵逆时针旋转

1.7 程序设计 4：命令行实现对 N 个整数排序

1.7.1 题目要求

编写一个 C 程序，通过命令行参数接收整数个数 N 和可选的排序标志-d，从标准输入读取 N 个整数，根据是否指定-d 参数进行升序或降序排序，最后输出排序结果。要求 n 个整数的存储无冗余。

1.7.2 设计程序流程图

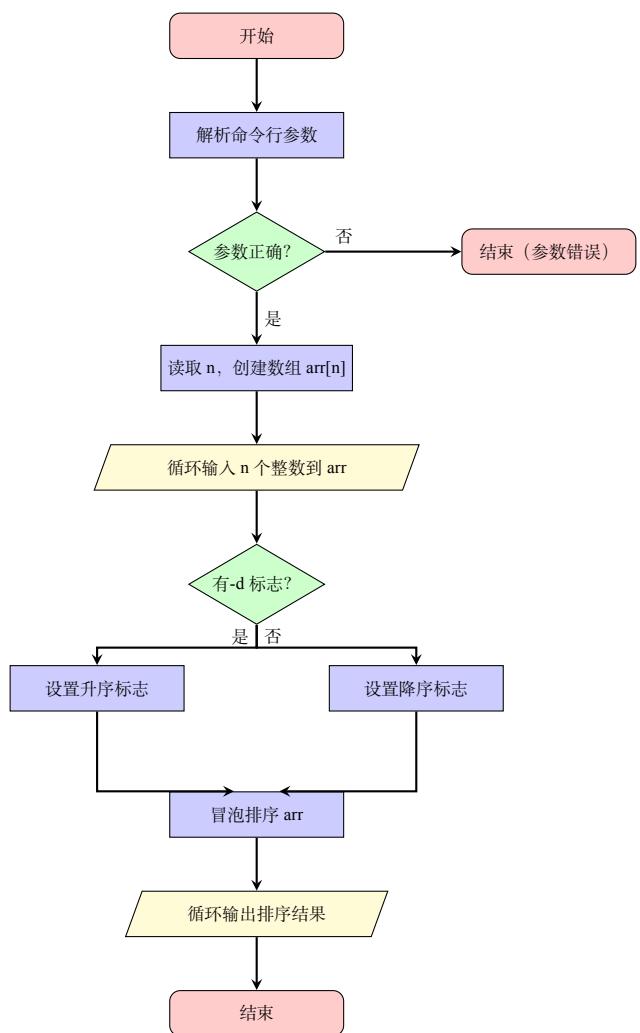


图 1-13 命令行整数排序程序流程图

1.7.3 完整程序代码

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 void sort(int *arr,int n,int is_desc){
4     int temp;
5     for(int i=0;i<n;i++){
6         for(int j=0;j<n-i-1;j++){
7             if((is_desc&&arr[j]<arr[j+1])||(!is_desc&&arr[j]>arr[j+1])){
8                 temp=arr[j];
9                 arr[j]=arr[j+1];
10                arr[j+1]=temp;
11            }
12        }
13    }
14 }
15 int main(int argc,char *argv[]){
16     if(argc<2){
17         printf("Usage:%s N [-d]\n",argv[0]);
18         return 1;
19     }
20     int n=atoi(argv[1]);
21     if(n<=0){
22         printf("Error:N must be positive integer\n");
23         return 1;
24     }
25     int arr[n];
26     for(int i=0;i<n;i++){
27         scanf("%d",&arr[i]);
28     }
29     int is_desc=0;
30     if(argc==3&&argv[2][0]=='-'&&argv[2][1]=='d'){
31         is_desc=1;
32     }
33     sort(arr,n,is_desc);
34     for(int i=0;i<n;i++){
35         printf("%d ",arr[i]);
36     }
37     printf("\n");
38     return 0;
39 }
```

Listing 10 命令行整数排序程序完整代码

1.7.4 程序运行示例



The screenshot shows a terminal window with two tabs: "测试结果" (Test Results) and "自测运行结果" (Self-test Execution Results). The "自测运行结果" tab is active, indicated by a green underline. It contains two sections: "自测输入" (Self-test Input) and "运行结果" (Execution Result). The input section shows the command `./sort 5 -d` followed by the numbers `4 3 8 5 1`. The result section shows the sorted output `8 5 4 3 1`.

```
测试结果      自测运行结果
自测输入          运行结果
./sort 5 -d
4 3 8 5 1
8 5 4 3 1
```

图 1-14 命令行实现对 N 个整数排序

1.8 程序设计 5：删除子串

1.8.1 题目要求

编写一个 C 程序，从主字符串中删除所有与给定子串相同的部分，并输出处理后的字符串以及是否进行了修改的标记。

1.8.2 设计程序流程图

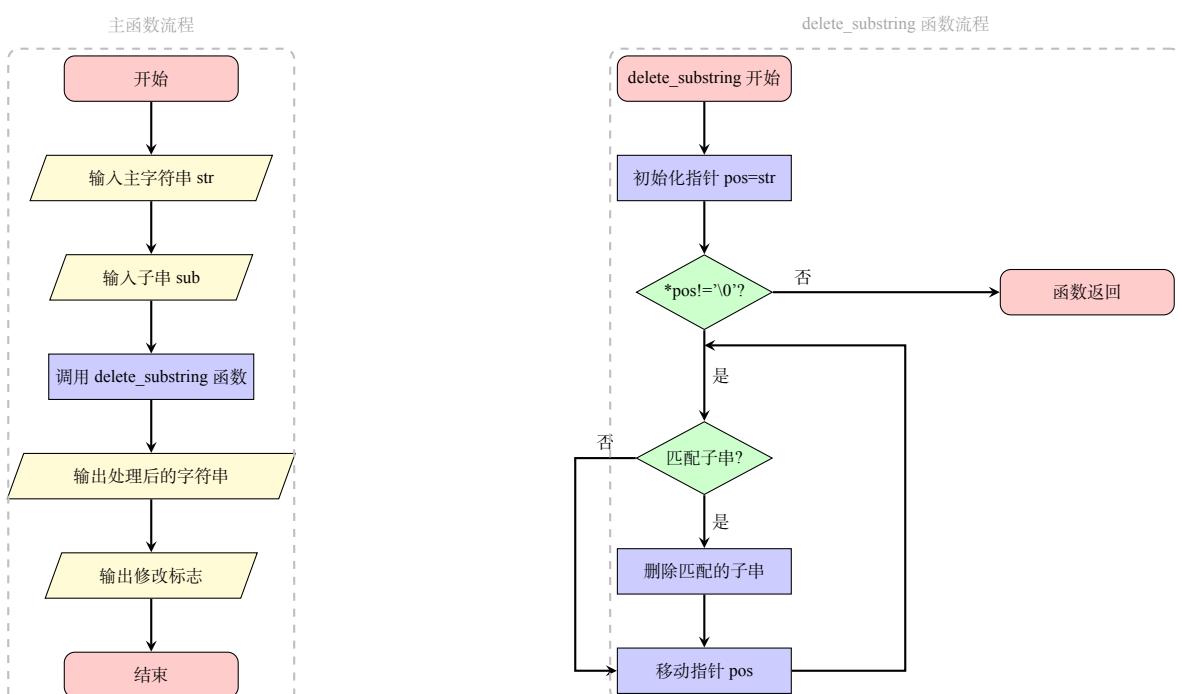


图 1-15 删除子串程序流程图

1.8.3 完整程序代码

```

1 #include<stdio.h>
2 #include<string.h>
3 void delete_substring(char *str,const char *sub){
4     char *pos;
5     int n=strlen(sub);
6     pos=str;
7     while(*pos]!='\0'){
8         int flag=1;
9         if((pos-str+n)>str+strlen(str)){
10             break;
11         }

```

```
12     for(int i=0;i<n;i++){
13         if(*(pos+i)!=*(sub+i)){
14             flag=0;
15             break;
16         }
17     }
18     if(flag==1){
19         memmove(pos,pos+n,strlen(pos+n)+1);
20         pos-=1;
21     }
22     pos+=1;
23 }
24 }
25 int main(){
26     char str[1000];
27     char sub[1000];
28     fgets(str,sizeof(str),stdin);
29     int n=strlen(str)-1;
30     fgets(sub,sizeof(sub),stdin);
31     str[strlen(str)-1]='\0';
32     delete_substring(str,sub);
33     printf("%s\n",str);
34     printf("%d",!(strlen(str)==n));
35     return 0;
36 }
```

Listing 11 删 除子串程序完整代码

1.8.4 程序运行示例

测试输入:	stay hungry stay foolish stay	—— 预期输出 ——	—— 实际输出 ——
	hungry foolish 1		hungry foolish 1

图 1-16 删 除字串

1.9 程序设计 6：非负整数积

1.9.1 题目要求

编写一个 C 程序，计算两个不超过 200 位的非负整数的乘积，使用高精度算法处理大数运算。

1.9.2 设计程序流程图

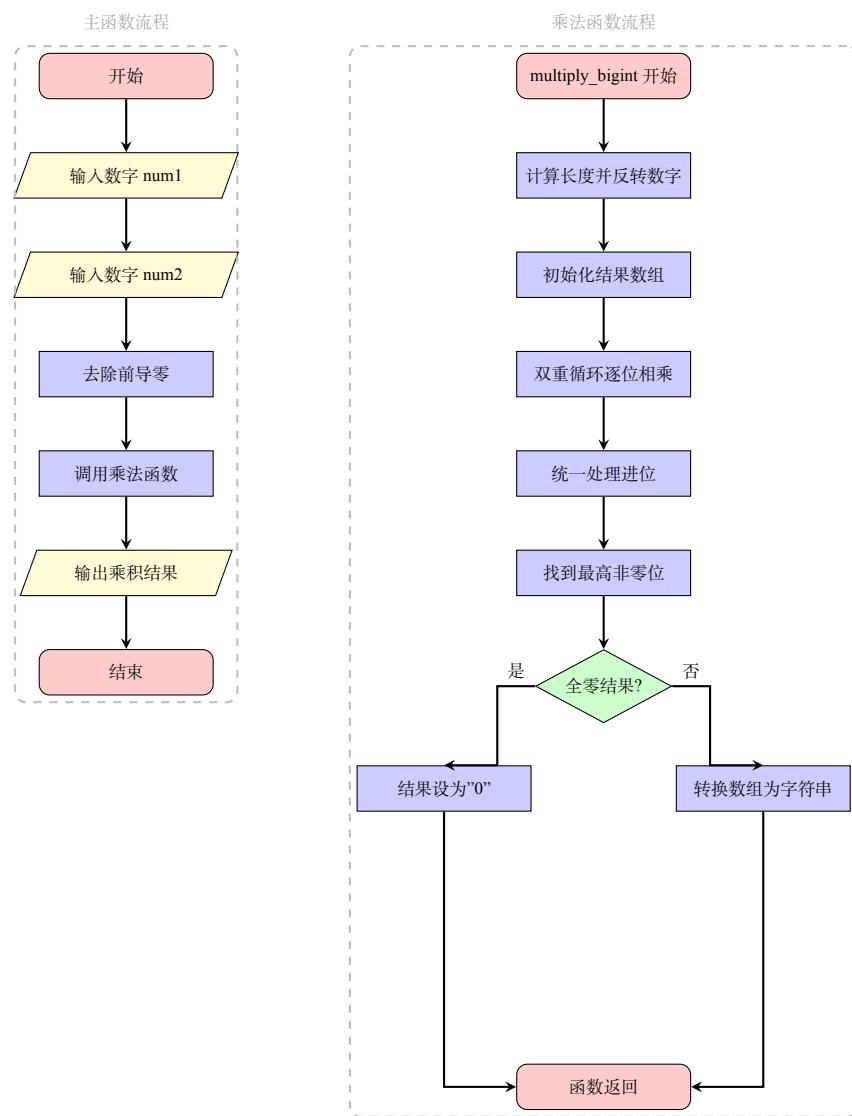


图 1-17 高精度整数乘法程序流程图

1.9.3 完整程序代码

```
1 #include<stdio.h>
2 #include<string.h>
3 #define MAX_LEN 400
4 void reverse(char *str){
5     int len=strlen(str);
6     for(int i=0;i<len/2;i++){
7         char temp=str[i];
8         str[i]=str[len-1-i];
9         str[len-1-i]=temp;
10    }
11 }
12 void multiply_bigint(char *num1,char *num2,char *result){
13     int len1=strlen(num1);
14     int len2=strlen(num2);
15     int result_len=len1+len2;
16     int res[MAX_LEN]={0};
17     reverse(num1);
18     reverse(num2);
19     for(int i=0;i<len1;i++){
20         for(int j=0;j<len2;j++){
21             res[i+j]+=(num1[i]-'0')*(num2[j]-'0');
22             if(res[i+j]>=10){
23                 res[i+j+1]+=res[i+j]/10;
24                 res[i+j]%=10;
25             }
26         }
27     }
28     for(int i=0;i<result_len;i++){
29         if(res[i]>=10){
30             res[i+1]+=res[i]/10;
31             res[i]%=10;
32         }
33     }
34     int highest=result_len-1;
35     while(highest>=0&&res[highest]==0){
36         highest--;
37     }
38     if(highest<0){
39         result[0]='0';
40         result[1]='\0';
41         return;
42     }
```

华中科技大学课程实验报告

```
43     int idx=0;
44
45     for(int i=highest;i>=0;i--){
46
47         result[idx++]=res[i]+'0';
48     }
49
50     result[idx]='\0';
51
52     int main(){
53
54         char num1[MAX_LEN/2+1];
55         char num2[MAX_LEN/2+1];
56         char result[MAX_LEN+1];
57
58         scanf("%s",num1);
59         scanf("%s",num2);
60
61         char *p1=num1;
62         char *p2=num2;
63
64         while(*p1=='0'&&*(p1+1)!='\0')p1++;
65         while(*p2=='0'&&*(p2+1)!='\0')p2++;
66
67         multiply_bigint(p1,p2,result);
68
69         printf("%s\n",result);
70
71         return 0;
72     }
```

Listing 12 高精度整数乘法程序完整代码

1.9.4 程序运行示例

图 1-18 高精度乘法

1.10 程序设计 7: 函数调度

1.10.1 题目要求

编写一个 C 程序，实现 8 个任务函数、一个调度函数和一个执行函数。调度函数使用最快的方式（函数指针数组）调度执行用户指定的任务函数。

1.10.2 设计程序流程图

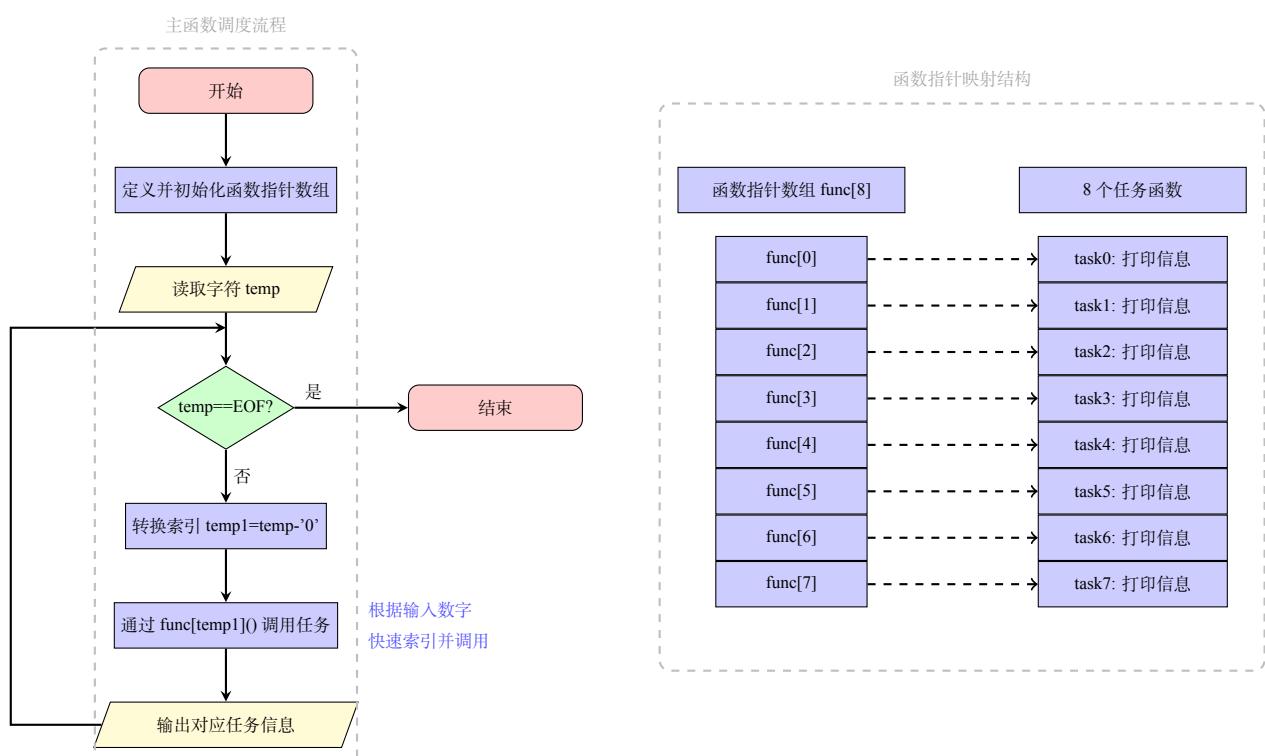


图 1-19 函数调度程序流程图

1.10.3 完整程序代码

```

1 #include<stdio.h>
2 void task0(){
3     printf("task0 is called!\n");
4 }
5 void task1(){
6     printf("task1 is called!\n");
7 }
8 void task2(){
9     printf("task2 is called!\n");
10}

```

```
11 void task3(){
12     printf("task3 is called!\n");
13 }
14 void task4(){
15     printf("task4 is called!\n");
16 }
17 void task5(){
18     printf("task5 is called!\n");
19 }
20 void task6(){
21     printf("task6 is called!\n");
22 }
23 void task7(){
24     printf("task7 is called!\n");
25 }
26 int main(){
27     typedef void (*fun)();
28     fun func[8]={
29         task0,task1,task2,task3,task4,task5,task6,task7
30     };
31     char temp;
32     int temp1;
33     while((temp=getchar())!=EOF){
34         temp1=temp-'0';
35         func[temp1]();
36     }
37     return 0;
38 }
```

Listing 13 函数调度程序完整代码

1.10.4 程序运行示例

```
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 1-20 函数调度

1.11 小结

本次实验全面深入地学习了 C 语言指针的各类应用，通过七个程序设计任务系统地掌握了指针的核心概念和高级用法。

1.11.1 主要收获与成果

1. 指针基础与调试技能

- 通过程序改错任务，深入理解了指针初始化的重要性
- 掌握了使用 DEV C++ 进行指针调试的基本方法
- 认识到未初始化指针导致的段错误及其调试技巧

2. 指针与字符串处理

- 实现了字符串拷贝函数的正确版本，理解了指针移动对返回值的影响
- 掌握了指针数组排序的两种实现方式（下标法和纯指针法）
- 学习了函数指针数组（转移表）替代 switch 语句的优化方法

3. 指针高级应用

- 通过指针操作实现整数字节分解，深入理解了内存存储结构
- 使用指针完成有序序列去重算法，掌握了指针与数组的配合使用
- 实现了矩阵旋转算法，熟练运用指针进行二维数组操作

4. 系统级编程能力

- 通过命令行参数处理程序，掌握了 argc/argv 的使用方法
- 实现子串删除功能，深入理解了字符串指针操作和内存移动
- 完成高精度乘法算法，提升了处理大数运算的能力

5. 函数调度机制

- 设计并实现了基于函数指针数组的任务调度系统
- 理解了函数指针作为回调机制的核心原理

1.11.2 技术难点与突破

- **二级指针在字符串排序中的应用：**通过双指针操作实现无下标引用的排序算法

- **指针运算与边界条件的精确控制:** 在处理内存移动和字符串操作时确保不越界
- **函数指针数组的初始化与调用机制:** 掌握静态函数指针数组的定义和使用
- **高精度算法的指针优化实现:** 通过指针操作提高大数运算效率

1.11.3 总结与反思

本次实验从基础到高级，系统性地训练了指针编程能力。从最开始的指针初始化错误调试，到复杂的函数指针调度系统，每一步都加深了对 C 语言内存管理和指针机制的理解。特别是在实现高精度乘法和命令行参数处理时，不仅巩固了指针操作技能，还培养了系统化编程思维。通过这次实验，认识到指针作为 C 语言核心特性，既是强大的工具，也需要谨慎使用。在未来的编程实践中，将更加注重指针的安全性检查，并尝试将学到的指针技巧应用于更复杂的数据结构和算法实现中。

2 实验 7 结构与联合

2.1 表达式求值的程序验证

2.1.1 题目要求

分析给定程序中的 6 个表达式，求值后通过编程验证结果的正确性。

2.1.2 初始状态分析

程序初始化以下变量：

- 字符数组 `u[] = "UVWXYZ"`, 包含字符: '`U`', '`V`', '`W`', '`X`', '`Y`', '`Z`', '`\0`'
- 字符数组 `v[] = "xyz"`, 包含字符: '`x`', '`y`', '`z`', '`\0`'
- 结构体类型 `T` 包含三个成员: `int x`、`char c`、`char *t`
- 结构体数组 `a[]` 包含两个元素:
 - `a[0] = {11, 'A', u}`: `x=11`, `c='A'`, `t` 指向 `u` 的首地址
 - `a[1] = {100, 'B', v}`: `x=100`, `c='B'`, `t` 指向 `v` 的首地址
- 结构体指针 `p = a`, 初始指向 `a[0]`

2.1.3 表达式分析与计算结果

1. Case 1: `(++p)->x`

- **求值过程:** 前缀递增运算符 `++` 先使指针 `p` 自增, 指向 `a[1]`, 然后通过箭头运算符访问成员 `x`
- **计算结果:** `a[1].x` 的值为 100
- **执行后状态:** `p` 指向 `a[1]`

2. Case 2: `p++; printf("%c", p->c);`

- **求值过程:** 后置递增运算符 `p++` 先使用 `p` 的当前值, 然后 `p` 自增指向 `a[1]`, 再通过箭头运算符访问成员 `c`
- **计算结果:** `a[1].c` 的值为 '`B`'
- **执行后状态:** `p` 指向 `a[1]`

3. Case 3: `*p++->t; printf("%c", *p->t);`

- **求值过程:** 运算符优先级: `->` 高于 `++`。先计算 `p->t` (得到 `a[0].t`)，然后 `p` 自增指向 `a[1]`。第一条语句解引用得到 '`U`' 但未使用。第二条语句访问 `p->t` 并解引用
- **计算结果:** `*p->t` 解引用 `a[1].t` 得到 '`x`'
- **执行后状态:** `p` 指向 `a[1]`，`a[0].t` 不变

4. Case 4: `*(++p)->t`

- **求值过程:** 先执行 `++p` 使 `p` 指向 `a[1]`，然后访问 `t` 成员，最后解引用
- **计算结果:** 解引用 `a[1].t` 得到 '`x`'
- **执行后状态:** `p` 指向 `a[1]`

5. Case 5: `++p->t`

- **求值过程:** 运算符优先级: `->` 高于 `++`。先计算 `p->t` (`a[0].t`)，然后对该指针执行前缀递增，使其指向 `u[1]`，最后解引用
- **计算结果:** 解引用得到 '`V`'
- **执行后状态:** `p` 仍指向 `a[0]`，但 `a[0].t` 现在指向 `u[1]` (原字符串的第二个字符)

6. Case 6: `++*p->t`

- **求值过程:** 先计算 `p->t` (`a[0].t`)，解引用得到字符 '`U`'，然后对该字符执行前缀递增
- **计算结果:** '`U`' 的 ASCII 值加 1 得到 '`V`'
- **执行后状态:** `p` 仍指向 `a[0]`，`u[0]` 的值被修改为 '`V`'

2.1.4 验证程序代码

```
1 #include<stdio.h>
2 int main(){
3     char u [] = "UVWXYZ", v [] = "xyz";
4     struct T{
5         int x;
6         char c;
7         char *t;
8     };
9     a [] = {{11, 'A', u}, {100, 'B', v}}, *p=a;
10    int temp;
11    scanf ("%d", &temp);
```

```
12     switch(temp){  
13         case 1:  
14             printf("%d", (++p)->x);  
15             break;  
16         case 2:  
17             p++;  
18             printf("%c", p->c);  
19             break;  
20         case 3:  
21             *p++=>t;  
22             printf("%c", *p->t);  
23             break;  
24         case 4:  
25             printf("%c", *(++p)->t);  
26             break;  
27         case 5:  
28             printf("%c", *++p->t);  
29             break;  
30         case 6:  
31             printf("%c", ++*p->t);  
32             break;  
33     }  
34     return 0;  
35 }
```

Listing 14 结构体指针表达式验证程序

2.1.5 验证结果说明

运行验证程序，输入 1-6 的测试用例编号，可以得到以下验证结果：

1. 测试用例 1：输出 100
2. 测试用例 2：输出 'B'
3. 测试用例 3：输出 'x'
4. 测试用例 4：输出 'x'
5. 测试用例 5：输出 'V'
6. 测试用例 6：输出 'V'

所有验证结果与分析预期完全一致，证明了表达式分析的正确性。

2.2 源程序修改替换（一）

2.2.1 原程序问题分析

原程序的目标是：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。链表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。

原程序存在以下问题：

1. **参数传递错误**: `create_list` 函数接受的是 `struct s_list *headp`, 即指针的值传递。在函数内部修改 `headp` 不会影响 `main` 函数中的 `head` 指针。
2. **输入方式固定**: 数组 `s` 是固定的，不能动态从键盘输入。
3. **逻辑错误**: `if(p[0]==0);` 语句后面有分号，导致即使第一个元素为 0 也会继续执行后续代码。
4. **未初始化**: 当第一个元素为 0 时，函数没有正确设置头指针。

2.2.2 原程序代码（错误标注）

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct s_list{
4     int data;
5     struct s_list *next;
6 };
7 void create_list(struct s_list *headp,int *p);
8 int main(void){
9     struct s_list *head=NULL,*p;
10    int s[]={1,2,3,4,5,6,7,8,0};
11    create_list(head,s);
12    p=head;
13    while (p){
14        printf("%d\t",p->data);
15        p=p->next;
16    }
17    printf("\n");
18    return 0;
19 }
20 void create_list(struct s_list *headp,int *p){
21     struct s_list *loc_head=NULL,*tail;
```

```
22     if(p[0]==0);
23 else {
24     loc_head=(struct s_list *)malloc(sizeof(struct s_list));
25     loc_head->data=*p++;
26     tail=loc_head;
27     while (*p)
28     {
29         tail->next=(struct s_list *)malloc(sizeof(struct s_list));
30         tail=tail->next;
31         tail->data=*p++;
32     }
33     tail->next=NULL;
34
35 }
36     headp=loc_head;
37 }
```

Listing 15 原程序（错误已标注）

1. **错误 1：**函数调用时传递 `head` 指针的值，而不是指针的地址，导致函数内部无法修改 `main` 函数中的 `head` 指针。
 2. **错误 2：**`if(p[0]==0);` 语句后面有分号，导致空语句，即使第一个元素为 0 也不会执行任何操作，且没有设置头指针。
 3. **错误 3：**函数内部修改的是局部变量 `headp`，这个修改不会影响 `main` 函数中的 `head` 指针。
 4. **额外需完善的问题：**数组 `s` 是固定的，无法动态输入。题目要求通过键盘输入给数组 `s` 的元素赋值，以 0 结束输入，且 0 作为数组的最后一个元素。
-
1. **错误 1：**函数调用时传递 `head` 指针的值，而不是指针的地址，导致函数内部无法修改 `main` 函数中的 `head` 指针。
 2. **错误 2：**`if(p[0]==0);` 语句后面有分号，导致空语句，即使第一个元素为 0 也不会执行任何操作，且没有设置头指针。
 3. **错误 3：**函数内部修改的是局部变量 `headp`，这个修改不会影响 `main` 函数中的 `head` 指针。
 4. **额外问题：**数组 `s` 是固定的，无法动态输入。

2.2.3 设计改进程序流程图

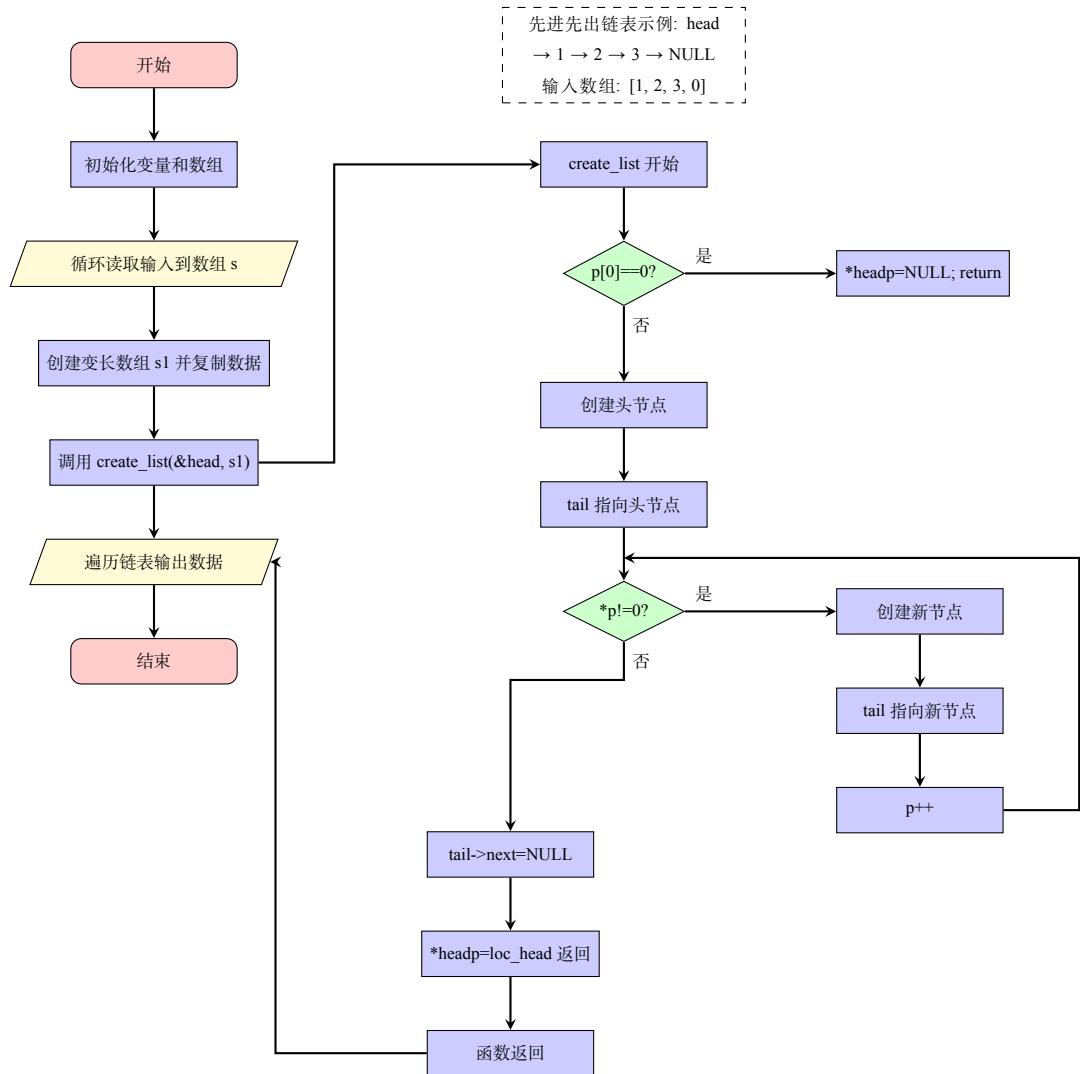


图 2-1 改进后的先进先出链表创建程序流程图

2.2.4 改进方案

针对上述问题，提出以下改进方案：

- 使用二级指针：**将 `create_list` 函数的参数改为 `struct s_list **headp`，这样可以在函数内部修改 `main` 函数中的头指针。
- 动态输入：**通过键盘输入数组元素，以 0 结束输入，且 0 作为数组的最后一个元素。
- 改进链表创建逻辑：**当输入数组第一个元素就是 0 时，直接设置头指针为 `NULL` 并返回。

4. 移除错误分号：修正 if 语句的逻辑。

2.2.5 改进后的完整程序代码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct s_list{
4     int data;
5     struct s_list *next;
6 };
7 void create_list(struct s_list **headp,int *p);
8 int main(void){
9     int s[100];
10    struct s_list *head=NULL,*p;
11    int top=0;
12    int temp;
13    while(scanf("%d", &temp) == 1){
14        s[top] = temp;
15        top += 1;
16    }
17    int s1[top];
18    for(int i=0;i<top;i++){
19        s1[i]=s[i];
20    }
21    create_list(&head,s1);
22    p=head;
23    while (p){
24        printf("%d\t",p->data);
25        p=p->next;
26    }
27    printf("\n");
28    return 0;
29 }
30 void create_list(struct s_list **headp,int *p){
31     struct s_list *loc_head=NULL,*tail;
32     if(p[0]==0){
33         *headp=NULL;
34         return;
35     }
36     else {
37         loc_head=(struct s_list *)malloc(sizeof(struct s_list));
38         loc_head->data=*p++;
39         tail=loc_head;
40         while (*p)
```

```
41     {
42         tail->next=(struct s_list *)malloc(sizeof(struct s_list));
43         tail=tail->next;
44         tail->data=*p++;
45     }
46     tail->next=NULL;
47
48 }
49 *headp=loc_head;
50 }
```

Listing 16 改进后的先进先出链表创建程序

2.2.6 程序运行示例

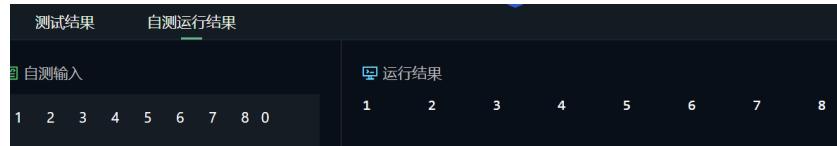


图 2-2 源程序修改替换（一）

2.3 源程序修改替换 (二)

2.3.1 题目要求

在原有先进先出链表程序基础上，修改为创建后进先出（LIFO）链表。新节点应插入到链表头部，使得最后输入的元素最先输出。

2.3.2 改进后的完整程序流程图

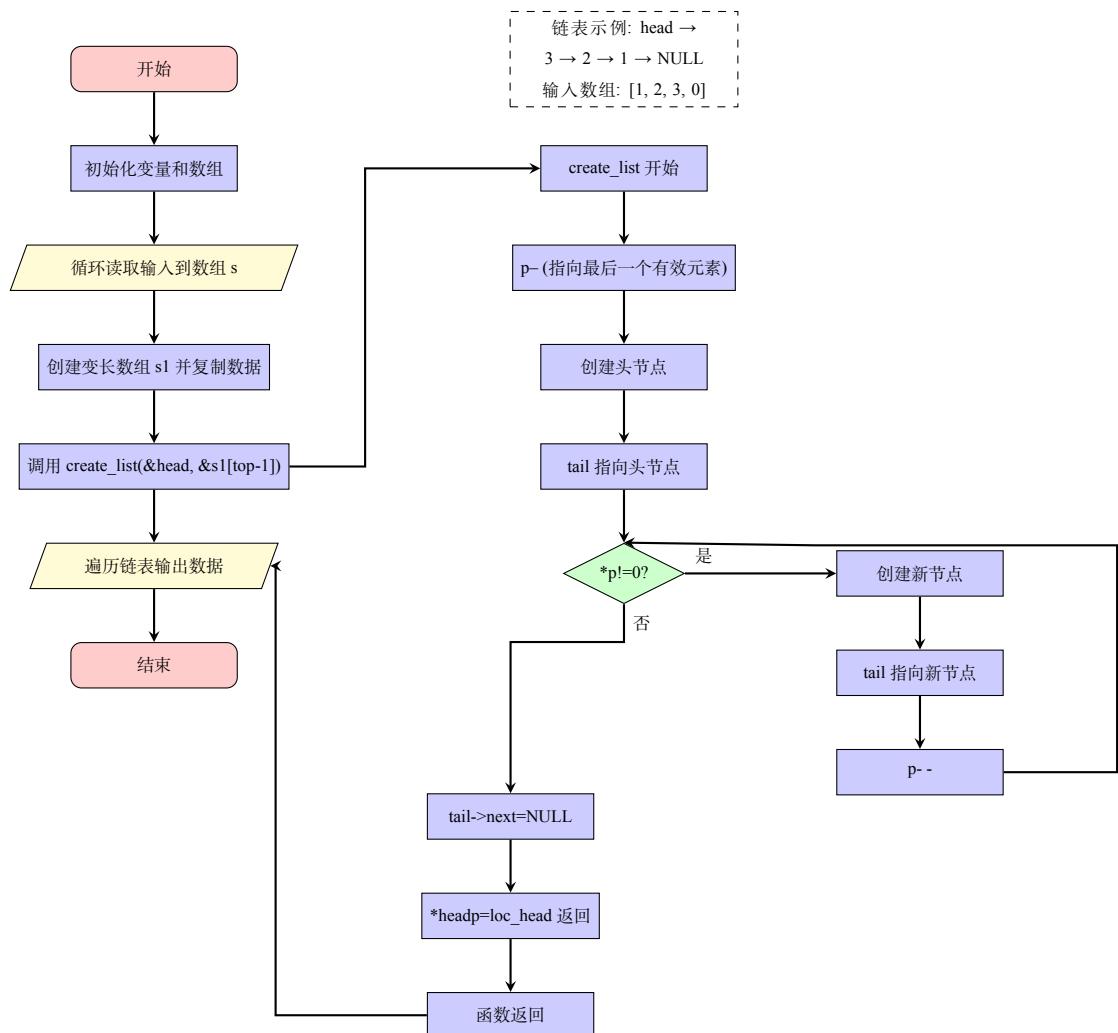


图 2-3 改进后的后进先出链表创建程序流程图

2.3.3 改进方案

为实现后进先出特性，采用以下改进方案：

1. 反向遍历数组：从数组末尾开始向前遍历，直到遇到 0 结束。

2. 头插法创建链表：将新节点插入到链表头部，保持后进先出特性。
3. 调整函数参数：传递数组最后一个元素的地址，便于反向遍历。

2.3.4 改进后的完整程序代码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct s_list{
4     int data;
5     struct s_list *next;
6 };
7 void create_list(struct s_list **headp,int *p);
8 int main(void){
9     int s[100];
10    struct s_list *head=NULL,*p;
11    int top=0;
12    int temp;
13    while(scanf("%d", &temp) == 1){
14        s[top] = temp;
15        top += 1;
16    }
17    int s1[top];
18    for(int i=0;i<top;i++){
19        s1[i]=s[i];
20    }
21    create_list(&head,&s1[top-1]);
22    p=head;
23    while (p){
24        printf("%d\t",p->data);
25        p=p->next;
26    }
27    printf("\n");
28    return 0;
29 }
30 void create_list(struct s_list **headp,int *p){
31     struct s_list *loc_head=NULL,*tail;
32     p--;
33     if(1){
34         loc_head=(struct s_list *)malloc(sizeof(struct s_list));
35         loc_head->data=*p;
36         tail=loc_head;
37         while (*p)
38         {
```

```
39         tail->next=(struct s_list *)malloc(sizeof(struct s_list));
40         tail=tail->next;
41         tail->data=*p--;
42     }
43     tail->next=NULL;
44
45 }
46 *headp=loc_head;
47 }
```

Listing 17 改进后的后进先出链表创建程序

2.3.5 程序运行示例



图 2-4 源程序修改替换（二）

2.4 程序设计 1：设计字段结构

2.4.1 题目要求

将一个 8 位无符号字节从最低位到最高位声明为 8 个字段，依次为 bit0, bit1,..., bit7，同时设计 8 个函数，第 i 个函数以 biti(i=0, 1, ..., 7) 为参数，并且在函数体内输出 biti 的值。将 8 个函数的名字存入一个函数指针数组 p_fun。如果 bit0 为 1，调用 p_fun[0] 指向的函数。如果 struct bits 中有多位为 1，则根据优先级从高到低顺序依次调用函数指针数组 p_fun 中相应元素指向的函数。

2.4.2 设计程序流程图

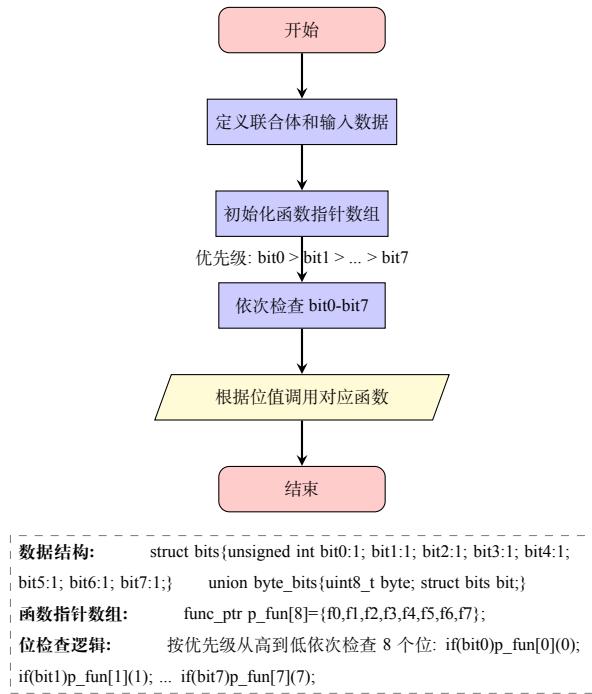


图 2-5 位字段结构与函数指针调度程序流程图

2.4.3 完整程序代码

```

1 #include <stdio.h>
2 #include <stdint.h>
3 struct bits {
4     unsigned int bit0 : 1;
5     unsigned int bit1 : 1;
6 /*此处省略6个定义*/
7 };

```

```
8 union byte_bits {
9     uint8_t byte;
10    struct bits bit;
11};
12 void f0(int b){printf("the function 0 is called!\n");}
13 /*此处省略7个声明*/
14 typedef void (*func_ptr)(int);
15 int main(){
16     union byte_bits data;
17     unsigned int input;
18     scanf("%u",&input);
19     data.byte=(uint8_t)(input&0xFF);
20     func_ptr p_fun[8]={f0,f1,f2,f3,f4,f5,f6,f7};
21     if(data.bit.bit0)p_fun[0](0);
22     if(data.bit.bit1)p_fun[1](1);
23 /*此处省略6个调用*/
24     return 0;
25 }
```

Listing 18 位字段结构与函数指针调度程序完整代码

2.4.4 程序运行示例

```
111
the function 0 is called!
the function 1 is called!
the function 2 is called!
the function 3 is called!
the function 5 is called!
the function 6 is called!

-----
Process exited after 1.617 seconds with return value 0
请按任意键继续. . . |
```

图 2-6 设计字段结构

2.5 程序设计 2：班级成绩单

2.5.1 题目要求

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 4 门课程的成绩。实现以下功能，并提供菜单选项：0. 退出 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. 统计每个学生的平均成绩（保留 2 位小数）5. 输出各位学生的学号、姓名、4 门课程的总成绩和平均成绩

2.5.2 设计程序流程图

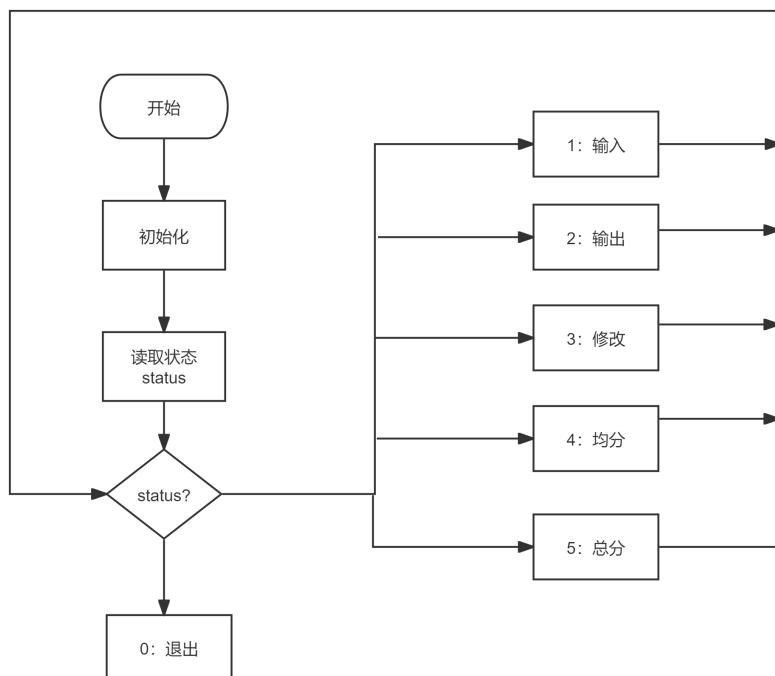


图 2-7 主函数流程图

下文将分别给出各状态流程图，其中状态 5 类似状态 2，此处不再赘述。

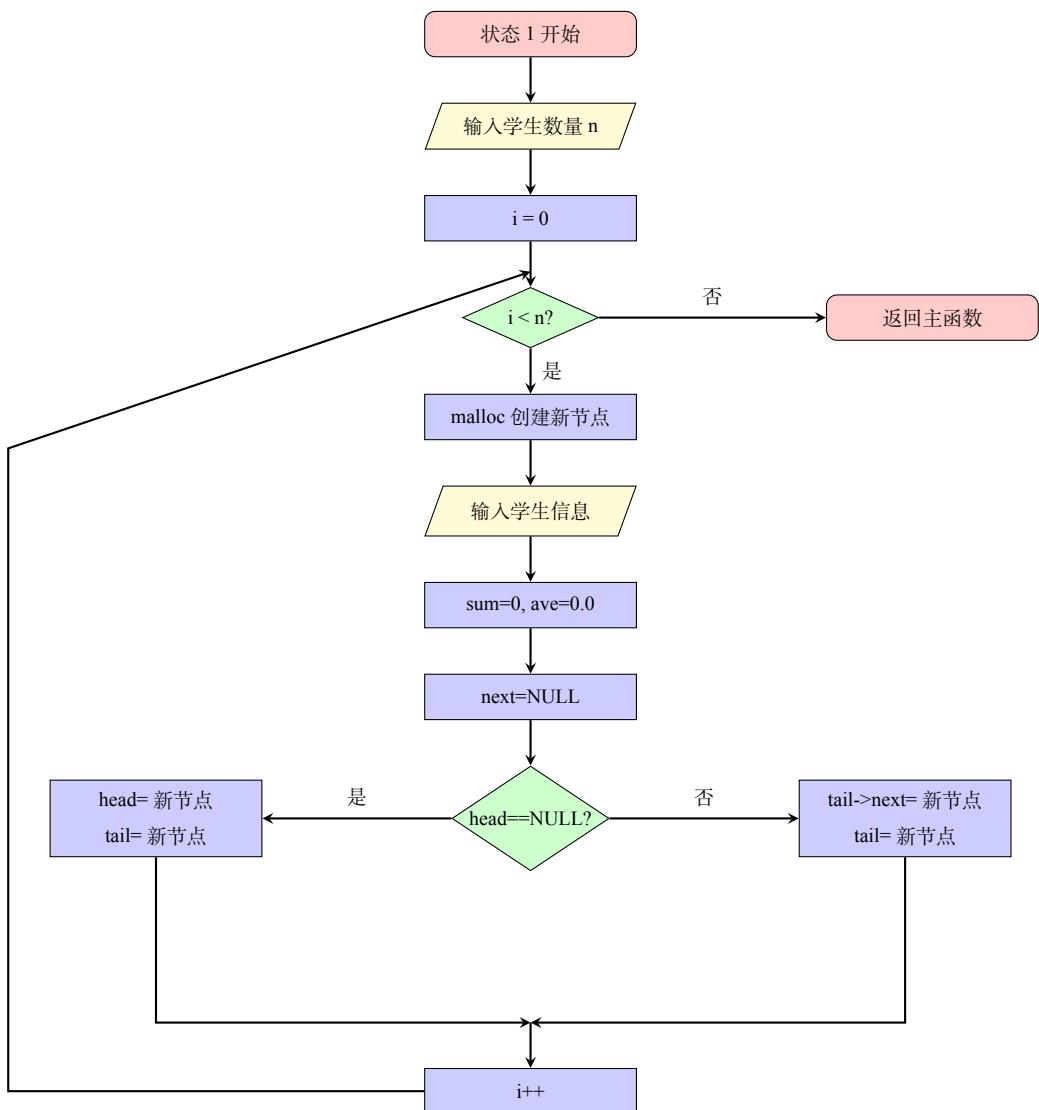


图 2-8 状态 1：输入学生信息流程图（链表实现）

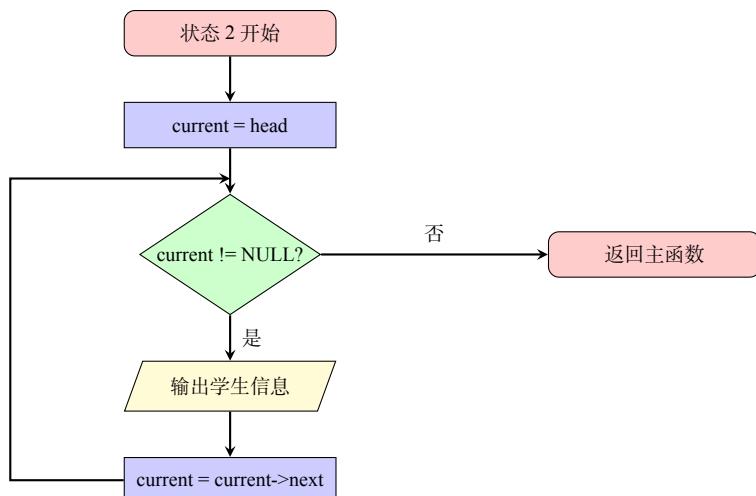


图 2-9 状态 2：输出学生信息流程图（链表实现）

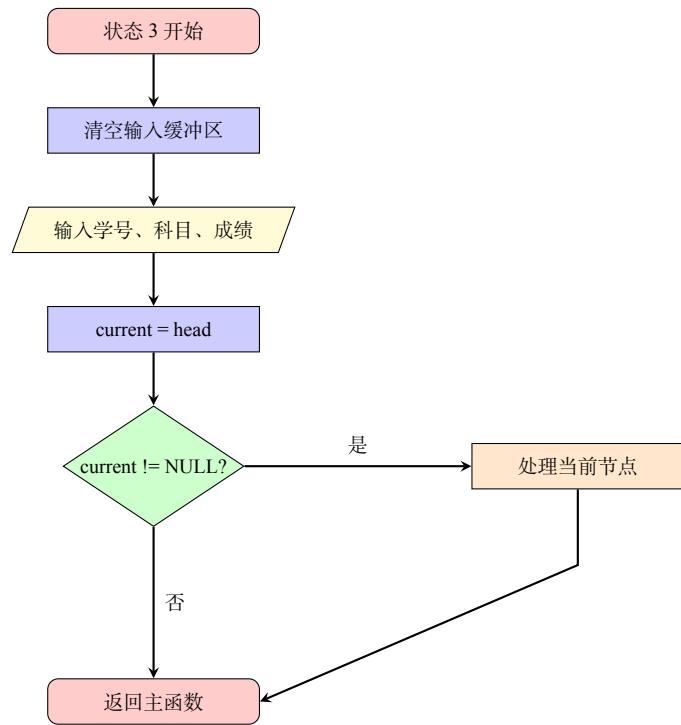


图 2-10 状态 3：修改学生成绩流程图（链表实现）

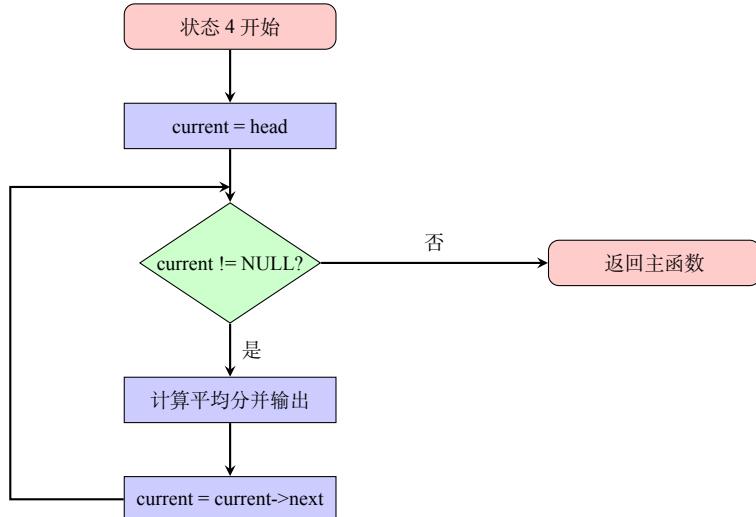


图 2-11 状态 4：计算平均成绩流程图（链表实现）

2.5.3 完整程序代码

```

1 #include<stdio.h>
2 #include<string.h>
3 #include<stdlib.h>
4 struct student {
5     char num[11];
6     char name[11];
7     int score[4];
8     int sum;
  
```

华中科技大学课程实验报告

```
9     float ave;
10    struct student *next;
11 };
12 int main() {
13     int status;
14     struct student *head = NULL;
15     struct student *tail = NULL;
16     int n;
17     while(1) {
18         scanf("%d", &status);
19         if(status == 0) {
20             break;
21         }
22         if(status == 1) {
23             scanf("%d", &n);
24             for(int i = 0; i < n; i++) {
25                 struct student *new_student = (struct
26                     student*)malloc(sizeof(struct student));
27                 scanf("%s %s %d %d %d %d",
28                     new_student->num,
29                     new_student->name,
30                     &new_student->score[0],
31                     &new_student->score[1],
32                     &new_student->score[2],
33                     &new_student->score[3]);
34                 new_student->sum = 0;
35                 new_student->ave = 0.0;
36                 new_student->next = NULL;
37                 if(head == NULL) {
38                     head = new_student;
39                     tail = new_student;
40                 } else {
41                     tail->next = new_student;
42                     tail = new_student;
43                 }
44             }
45             if(status == 2) {
46                 struct student *current = head;
47                 while(current != NULL) {
48                     printf("%s %s %d %d %d %d\n",
49                         current->num,
50                         current->name,
51                         current->score[0],
52                         current->score[1],
```

华中科技大学课程实验报告

```
53         current->score[2],
54         current->score[3]);
55     current = current->next;
56 }
57 }
58 if(status == 3) {
59     while(getchar() != '\n');
60     char tnum[11];
61     int dis;
62     int tscore;
63     scanf("%s %d %d", tnum, &dis, &tscore);
64     struct student *current = head;
65     while(current != NULL) {
66         if(strcmp(current->num, tnum) == 0) {
67             if(dis >= 1 && dis <= 4) {
68                 current->score[dis - 1] = tscore;
69             }
70             break;
71         }
72         current = current->next;
73     }
74 }
75 if(status == 4) {
76     struct student *current = head;
77     while(current != NULL) {
78         current->sum = 0;
79         for(int j = 0; j < 4; j++) {
80             current->sum += current->score[j];
81         }
82         current->ave = current->sum / 4.0;
83         printf("%s %s %.2f\n",
84                current->num,
85                current->name,
86                current->ave);
87         current = current->next;
88     }
89 }
90 if(status == 5) {
91     struct student *current = head;
92     while(current != NULL) {
93         if(current->sum == 0) {
94             current->sum = 0;
95             for(int j = 0; j < 4; j++) {
96                 current->sum += current->score[j];
97             }
98         }
99     }
100 }
```

```
98         current->ave = current->sum / 4.0;
99     }
100    printf("%s %s %d %.2f\n",
101           current->num,
102           current->name,
103           current->sum,
104           current->ave);
105    current = current->next;
106 }
107 }
108 }
109 struct student *current = head;
110 while(current != NULL) {
111     struct student *temp = current;
112     current = current->next;
113     free(temp);
114 }
115 return 0;
116 }
```

Listing 19 班级成绩单管理系统完整代码

2.5.4 程序运行示例

```
1
2
U202512345 Jack 99 100 80 96
U202554321 Rose 89 94 85 100
2
U202512345 Jack 99 100 80 96
U202554321 Rose 89 94 85 100
3
U202554321 1 66
4
U202512345 Jack 93.75
U202554321 Rose 86.25
5
U202512345 Jack 375 93.75
U202554321 Rose 345 86.25
0

-----
Process exited after 66.15 seconds with return value 0
请按任意键继续. . . |
```

图 2-12 班级成绩单

2.6 程序设计 3：成绩排序（一）

2.6.1 题目要求

在原有班级成绩单管理系统（链表实现）的基础上，增加按照平均成绩进行升序排序的功能。

2.6.2 设计函数流程图

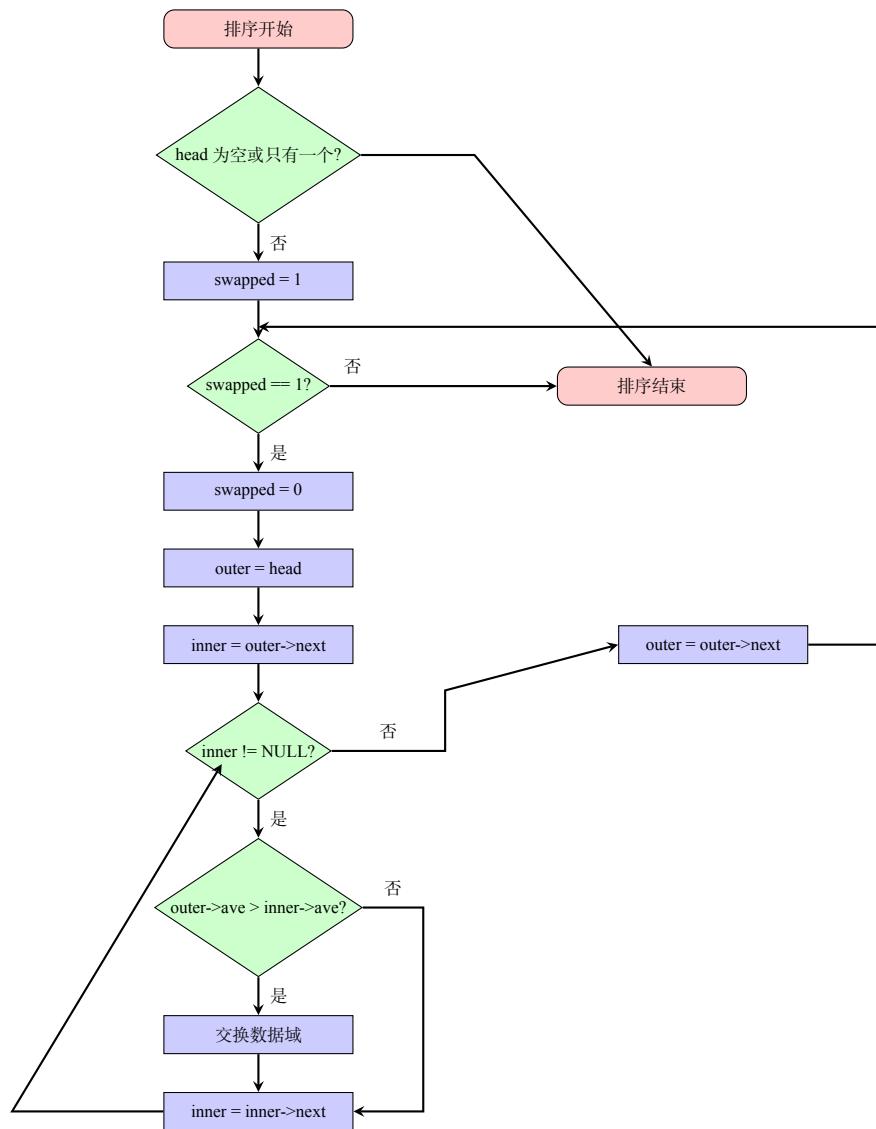


图 2-13 按平均成绩升序排序流程图（冒泡法）

安插位置说明

1. 状态 2：在输出学生信息前，先确保所有学生都有平均成绩，然后排序
2. 状态 4：在计算并输出平均成绩前，先排序

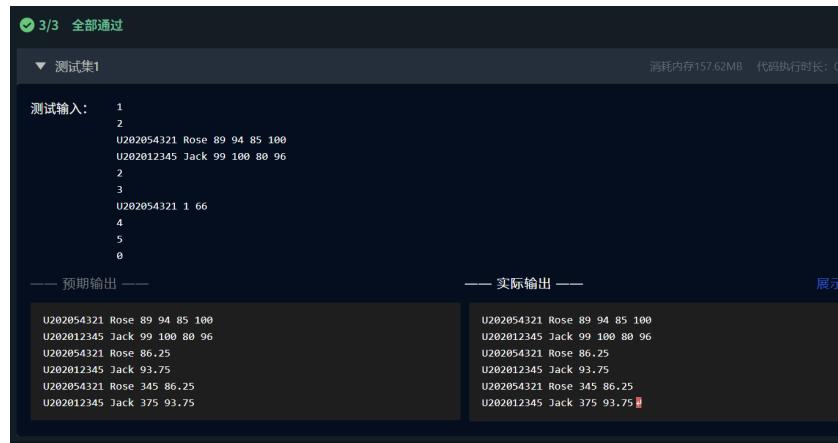
3. 状态 5：先确保所有学生都有平均成绩，然后排序再输出

2.6.3 排序函数代码

```
1 void swapStudentData(struct student *a, struct student *b) {
2     struct student temp;
3     strcpy(temp.num, a->num);
4     strcpy(temp.name, a->name);
5     for(int i = 0; i < 4; i++) temp.score[i] = a->score[i];
6     temp.sum = a->sum;
7     temp.ave = a->ave;
8     strcpy(a->num, b->num);
9     strcpy(a->name, b->name);
10    for(int i = 0; i < 4; i++) a->score[i] = b->score[i];
11    a->sum = b->sum;
12    a->ave = b->ave;
13    strcpy(b->num, temp.num);
14    strcpy(b->name, temp.name);
15    for(int i = 0; i < 4; i++) b->score[i] = temp.score[i];
16    b->sum = temp.sum;
17    b->ave = temp.ave;
18 }
19 void sortByAverage(struct student *head) {
20     if(head == NULL || head->next == NULL) {
21         return;
22     }
23     struct student *outer, *inner;
24     int swapped;
25     do {
26         swapped = 0;
27         outer = head;
28
29         while(outer->next != NULL) {
30             inner = outer->next;
31             if(outer->ave > inner->ave) {
32                 swapStudentData(outer, inner);
33                 swapped = 1;
34             }
35             outer = outer->next;
36         }
37     } while(swapped);
38 }
```

Listing 20 排序算法代码

2.6.4 程序运行示例



The screenshot shows a software interface for testing a program. At the top, it says "3/3 全部通过" (All passed) and "消耗内存157.62MB 代码执行时长: 0". Below this, there's a section titled "测试集1" (Test Set 1) with a dropdown arrow. Under "测试输入:" (Test Input), there is a list of student records:

```
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202054321 Rose 86.25
U202012345 Jack 93.75
U202054321 Rose 345 86.25
U202012345 Jack 375 93.75
```

Below the input, there are two sections: "预期输出" (Expected Output) and "实际输出" (Actual Output). Both sections show the same list of student records, indicating a successful run.

图 2-14 成绩排序（一）

2.7 程序设计 4：成绩排序（二）

2.7.1 题目要求

用交换结点指针域的方法升序排序

2.7.2 节点交换操作流程图

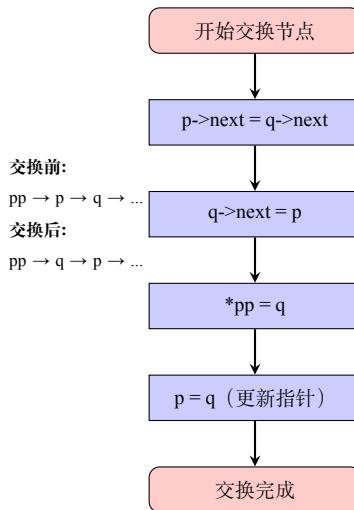


图 2-15 交换相邻节点指针域的详细操作

2.7.3 核心修改代码

```

1 void swapNodes(struct student **head, struct student *prev, struct
2   student *curr, struct student *next){
3     if(prev==NULL){
4       curr->next=next->next;
5       next->next=curr;
6       *head=next;
7     }else{
8       prev->next=next;
9       curr->next=next->next;
10      next->next=curr;
11    }
12  void sortByAverageSelection(struct student **head){
13    if(*head==NULL||(*head)->next==NULL) return;
14    struct student **pp,*p,**min_pp,*min_p;
15    struct student *q;
16    for(pp=head;*pp!=NULL;pp=&(*pp)->next){
  
```

```
17     min_pp=pp;
18     min_p=*pp;
19     for(q=(*pp)->next;q!=NULL;q=q->next){
20         if(q->ave<min_p->ave){
21             min_pp=&(*pp)->next;
22             min_p=q;
23         }
24     }
25     if(min_p!=*pp){
26         *min_pp=min_p->next;
27         min_p->next=*pp;
28         *pp=min_p;
29         pp=&min_p;
30     }
31 }
32 }
```

Listing 21 排序算法代码

2.7.4 程序运行示例

The screenshot shows a dark-themed programming environment interface. At the top, it says "3/3 全部通过" (All passed). Below that, there's a section titled "▼ 测试集1" (Test Set 1) with a "展示" (Show) button. On the left, under "测试输入:" (Test Input), there is a multi-line text area containing student IDs and their scores. On the right, under "—— 预期输出 ——" (Expected Output) and "—— 实际输出 ——" (Actual Output), there are two multi-line text areas showing the sorted list of students. The "Actual Output" matches the "Expected Output".

测试输入:

```
1
2
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
2
3
U202054321 1 66
4
5
0
```

—— 预期输出 ——

```
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202054321 Rose 86.25
U202012345 Jack 93.75
U202054321 Rose 345 86.25
U202012345 Jack 375 93.75
```

—— 实际输出 ——

```
U202054321 Rose 89 94 85 100
U202012345 Jack 99 100 80 96
U202054321 Rose 86.25
U202012345 Jack 93.75
U202054321 Rose 345 86.25
U202012345 Jack 375 93.75
```

图 2-16 成绩排序 (二)

2.8 程序设计 5：回文字符串判断

2.8.1 题目要求

输入一个任意长度的字符串，使用单链表存储该字符串，然后判断该字符串是否为回文字符串。

2.8.2 设计程序流程图

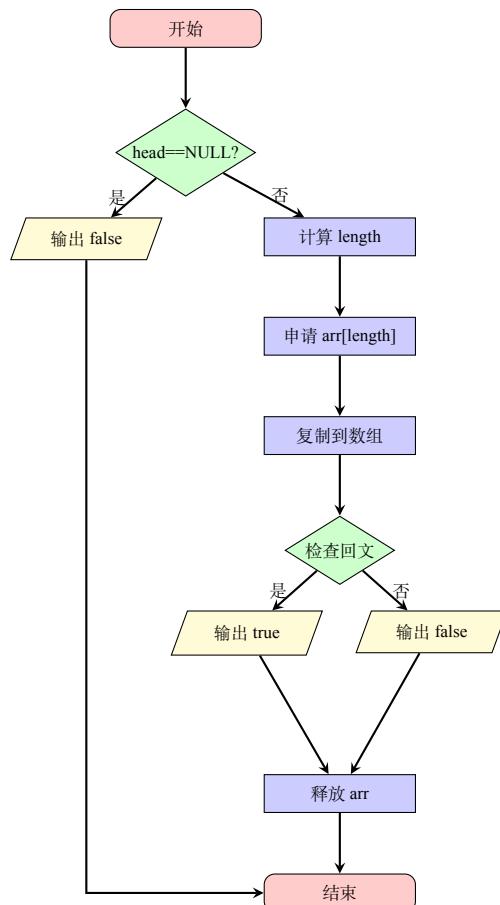


图 2-17 回文字符串判断算法流程图

2.8.3 完整程序代码

```

1
2 void createLinkList(C_NODE **headp, char s[]) {
3     *headp = NULL;
4     C_NODE *tail = NULL;
5
6     for(int i = 0; s[i] != '\0'; i++) {

```

华中科技大学课程实验报告

```
7     C_NODE *new_node = (C_NODE*)malloc(sizeof(C_NODE));
8     if(new_node == NULL) {
9         return;
10    }
11    new_node->data = s[i];
12    new_node->next = NULL;
13
14    if(*headp == NULL) {
15        *headp = new_node;
16        tail = new_node;
17    } else {
18        tail->next = new_node;
19        tail = new_node;
20    }
21 }
22
23
24 void judgePalindrome(C_NODE *head) {
25     if(head == NULL) {
26         printf("false\n");
27         return;
28     }
29
30     int length = 0;
31     C_NODE *current = head;
32     while(current != NULL) {
33         length++;
34         current = current->next;
35     }
36
37     char *arr = (char*)malloc(length * sizeof(char));
38     if(arr == NULL) {
39         printf("false\n");
40         return;
41     }
42
43     current = head;
44     for(int i = 0; i < length; i++) {
45         arr[i] = current->data;
46         current = current->next;
47     }
48
49     int is_palindrome = 1;
50     for(int i = 0; i < length / 2; i++) {
51         if(arr[i] != arr[length - 1 - i]) {
```

```
52         is_palindrome = 0;
53         break;
54     }
55 }
56
57 if(is_palindrome) {
58     printf("true\n");
59 } else {
60     printf("false\n");
61 }
62
63 free(arr);
64 }
```

Listing 22 排序算法代码

2.8.4 程序运行示例



图 2-18 回文字符串

2.9 程序设计 6：逆波兰表达式

2.9.1 题目要求

利用值栈对逆波兰表达式进行求值。逆波兰表达式从键盘输入，其中的运算符仅包含加、减、乘、除 4 种运算，表达式中的数都是十进制数，用换行符结束输入。由于逆波兰表达式的长度不限，所以值栈要用后进先出链表实现。

2.9.2 设计程序流程（由于分支较多，此处改用文字描述）

1. 数据结构初始化

1. 定义链表节点结构 Node，包含数据域 data 和指针域 next
2. 定义栈结构 Stack，包含栈顶指针 top
3. 实现栈操作函数：
 - initStack(): 初始化空栈
 - isEmpty(): 判断栈是否为空
 - push(): 元素入栈
 - pop(): 元素出栈并返回值
 - peek(): 获取栈顶元素值
 - freeStack(): 释放栈内存

2. 辅助函数定义

1. isOperator(): 判断字符是否为运算符 ('+', '- ', '*', '/')
2. performOperation(): 执行二元运算，根据运算符计算两个操作数的结果

3. 主算法流程

1. 初始化阶段：
 - 创建并初始化空栈
 - 初始化变量： num=0, readingNumber=false, isNegative=false
2. 字符读取循环：
 - 从标准输入读取一个字符 c
 - 如果 c 为 EOF，跳转到步骤 3.8

3. 数字字符处理:

- 如果 c 是数字字符 ('0'-'9'):
 - 更新 num = num × 10 + (c - '0')
 - 设置 readingNumber = true
 - 返回步骤 3.2 继续读取下一个字符

4. 减号特殊处理:

- 如果 c 是减号 '-':
 - (a) 预读下一个字符 nextChar
 - (b) 如果 nextChar 是数字:
 - 设置 isNegative = true
 - 设置 num = nextChar - '0'
 - 设置 readingNumber = true
 - 循环读取后续数字字符并累加到 num
 - 将最后一个非数字字符放回输入缓冲区
 - 返回步骤 3.2
 - (c) 否则 (nextChar 不是数字):
 - 将 nextChar 放回输入缓冲区
 - 如果 readingNumber 为真:
 - * 如果 isNegative 为真, 设置 num = -num
 - * 将 num 入栈
 - * 重置 num=0, readingNumber=false, isNegative=false
 - 从栈中弹出两个操作数 (先弹出 b, 再弹出 a)
 - 计算 result = a - b
 - 将 result 入栈
 - 返回步骤 3.2

5. 其他运算符处理:

- 如果 c 是 '+', '*', '/' 之一:
 - (a) 如果 readingNumber 为真:
 - 如果 isNegative 为真, 设置 num = -num
 - 将 num 入栈

- 重置 num=0, readingNumber=false, isNegative=false

(b) 从栈中弹出两个操作数 (先弹出 b, 再弹出 a)

(c) 根据运算符计算 result:

- '+': result = a + b

- '*': result = a × b

- '/': result = a ÷ b (整数除法)

(d) 将 result 入栈

(e) 返回步骤 3.2

6. 空格和制表符处理:

• 如果 c 是空格' ' 或制表符'\t':

- 如果 readingNumber 为真:

* 如果 isNegative 为真, 设置 num = -num

* 将 num 入栈

* 重置 num=0, readingNumber=false, isNegative=false

- 返回步骤 3.2

7. 换行符处理:

• 如果 c 是换行符'\n':

(a) 如果 readingNumber 为真:

- 如果 isNegative 为真, 设置 num = -num

- 将 num 入栈

- 重置 num=0, readingNumber=false, isNegative=false

(b) 检查栈状态:

- 如果栈不为空且栈中只有一个元素(stack.top != NULL && stack.top->next == NULL):

* 输出栈顶元素的值作为计算结果

* 跳转到步骤 3.9 结束程序

- 否则:

* 输出表达式错误信息

* 跳转到步骤 3.9 结束程序

8. 非法字符处理:

- 如果 c 是其他字符:
 - 释放栈内存
 - 返回错误状态码 1

9. 程序结束:

- 释放栈内存
- 返回成功状态码 0

2.9.3 完整程序代码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #include <stdbool.h>
5 typedef struct Node {
6     int data;
7     struct Node* next;
8 } Node;
9 typedef struct {
10     Node* top;
11 } Stack;
12 void initStack(Stack* stack) {
13     stack->top = NULL;
14 }
15 bool isEmpty(Stack* stack) {
16     return stack->top == NULL;
17 }
18 void push(Stack* stack, int value) {
19     Node* newNode = (Node*)malloc(sizeof(Node));
20     newNode->data = value;
21     newNode->next = stack->top;
22     stack->top = newNode;
23 }
24 int pop(Stack* stack) {
25     Node* temp = stack->top;
26     int value = temp->data;
27     stack->top = temp->next;
28     free(temp);
29     return value;
30 }
31 int peek(Stack* stack) {
32     return stack->top->data;
```

华中科技大学课程实验报告

```
33 }
34 void freeStack(Stack* stack) {
35     while (!isEmpty(stack)) {
36         pop(stack);
37     }
38 }
39 bool isOperator(char c) {
40     return c == '+' || c == '-' || c == '*' || c == '/';
41 }
42 int performOperation(int a, int b, char op) {
43     switch (op) {
44         case '+': return a + b;
45         case '-': return a - b;
46         case '*': return a * b;
47         case '/': return a / b;
48         default: return 0;
49     }
50 }
51 int main() {
52     Stack stack;
53     initStack(&stack);
54     char c;
55     int num = 0;
56     bool readingNumber = false;
57     bool isNegative = false;
58     while ((c = getchar()) != EOF) {
59         if (isdigit(c)) {
60             num = num * 10 + (c - '0');
61             readingNumber = true;
62         } else if (c == '-') {
63             char nextChar = getchar();
64             if (isdigit(nextChar)) {
65                 isNegative = true;
66                 num = (nextChar - '0');
67                 readingNumber = true;
68                 while (isdigit(nextChar = getchar())) {
69                     num = num * 10 + (nextChar - '0');
70                 }
71                 ungetc(nextChar, stdin);
72             } else {
73                 ungetc(nextChar, stdin);
74                 if (readingNumber) {
75                     if (isNegative) {
76                         num = -num;
77                         isNegative = false;
```

华中科技大学课程实验报告

```
78         }
79         push(&stack, num);
80         num = 0;
81         readingNumber = false;
82     }
83     int b = pop(&stack);
84     int a = pop(&stack);
85     int result = performOperation(a, b, '-');
86     push(&stack, result);
87 }
88 } else if (isOperator(c)) {
89     if (readingNumber) {
90         if (isNegative) {
91             num = -num;
92             isNegative = false;
93         }
94         push(&stack, num);
95         num = 0;
96         readingNumber = false;
97     }
98     int b = pop(&stack);
99     int a = pop(&stack);
100    int result = performOperation(a, b, c);
101    push(&stack, result);
102 } else if (c == ' ' || c == '\t') {
103     if (readingNumber) {
104         if (isNegative) {
105             num = -num;
106             isNegative = false;
107         }
108         push(&stack, num);
109         num = 0;
110         readingNumber = false;
111     }
112 } else if (c == '\n') {
113     if (readingNumber) {
114         if (isNegative) {
115             num = -num;
116             isNegative = false;
117         }
118         push(&stack, num);
119         num = 0;
120         readingNumber = false;
121     }
122     if (stack.top != NULL && stack.top->next == NULL) {
```

```
123         printf("%d\n", peek(&stack));
124         break;
125     } else {
126         freeStack(&stack);
127         return 1;
128     }
129 } else {
130     freeStack(&stack);
131     return 1;
132 }
133 }
134 freeStack(&stack);
135 return 0;
136 }
```

Listing 23 排序算法代码

2.9.4 程序运行示例

```
2 1 + 3 -
0
-----
Process exited after 5.808 seconds with return value 0
请按任意键继续. . . |
```

图 2-19 逆波兰表达式

2.10 小结

本实验重点学习了 C 语言中结构体、联合体及链表数据结构，通过六个程序设计任务掌握了复合数据类型的定义、使用和管理。

2.10.1 主要收获与成果

1. 结构体指针操作

- 通过表达式求值验证，深入理解了结构体指针的运算规则
- 掌握了箭头运算符与自增自减运算符的优先级关系
- 学会了通过指针访问结构体嵌套成员的方法

2. 链表设计与实现

- 实现了先进先出（FIFO）链表创建函数，理解了头指针传递机制
- 改进为后进先出（LIFO）链表，掌握了链表构建的不同策略
- 通过二级指针参数解决了头指针修改问题

3. 位字段与联合体

- 设计并实现了 8 位字节的位字段结构
- 掌握了联合体在类型转换和内存共享中的应用
- 实现了基于函数指针数组的位检查调度系统

4. 综合管理系统开发

- 设计并实现了完整的班级成绩单管理系统
- 掌握了链表在动态数据管理中的优势
- 实现了数据的增删改查及统计功能

5. 算法与数据结构

- 实现了两种链表排序算法：数据交换法和节点交换法
- 掌握了回文字符串的链表判断算法
- 设计并实现了逆波兰表达式求值器的栈结构

6. 系统设计能力

- 通过菜单驱动的交互设计，掌握了用户界面设计原则
- 实现了模块化的函数设计，提高了代码可维护性
- 掌握了动态内存管理的完整生命周期

2.10.2 技术难点与突破

- **二级指针在链表头节点修改中的应用：**通过传递头指针的地址，实现在函数内部修改头指针
- **位字段结构的内存布局与访问优化：**理解位字段的存储方式，实现高效的位置位操作
- **链表排序中相邻节点交换的指针操作：**掌握在不破坏链表结构的前提下交换节点
- **逆波兰表达式求值的栈实现与错误处理：**设计健壮的栈操作函数，处理各种边界情况

2.10.3 总结与反思

本实验系统性地学习了链表，以及相关的结构体和联合体。通过从简单的链表创建到复杂的成绩管理系统开发，逐步掌握了动态数据结构的构建、管理和优化方法。通过两种不同的链表排序算法实现，深入理解了指针操作的微妙之处。在实现逆波兰表达式求值器时，栈结构与链表的完美结合，展现了数据结构之间的协同工作能力。在未来的学习中，将尝试将链表等数据结构应用于更复杂的应用场景，并探索更多高效的数据组织方式。

参考文献

- [1] 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