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 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 程序改错与跟踪调试

1.1.1 原代码

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

```
    char s1[100];
    char s2[100];
    char *s3;
    scanf("%s",s2);
    strcopy(s1,s2);
    printf("%s\n",s1);
    scanf("%s",s2);
    s3=strcopy(s1,s2);
    printf("%s\n",s3);
    return 0;
}
char *strcopy(char *t,const char *s){
    char *start = t;
    while ((*t++ = *s++));
    return start;
}
```

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 strcopy 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

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/* 对指针数组s指向的size个字符串进行升序排序 */
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 N;
    scanf("%d",&N);
    N+=1;
    int i;
    char *s[N], t[50];
    for(i=0;i<N;i++)
    {
       gets(t);
       s[i] = (char *)malloc(strlen(t)+1);
       strcpy(s[i],t);
    }
    strsort(s,N);
    for(i=0;i<N;i++)  puts(s[i]);
    return 0;
}
```

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。

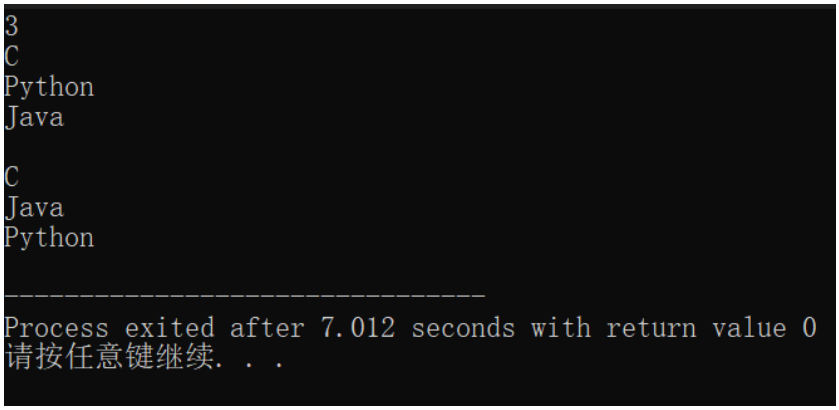


图 1-5　运行截图

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：

```
void strsort(char **s, int size)
{
    char **p, **q;
    char *temp;
        for(p = s; p < s + size - 1; p++) {
        for(q = s; q < s + size - 1 - (p - s); q++) {
            if(strcmp(*q, *(q + 1)) > 0) {
                temp = *q;
                *q = *(q + 1);
                *(q + 1) = temp;            }}}}
```

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

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
    char*(*p)(char a[],char b[]);
    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;
}
```

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。

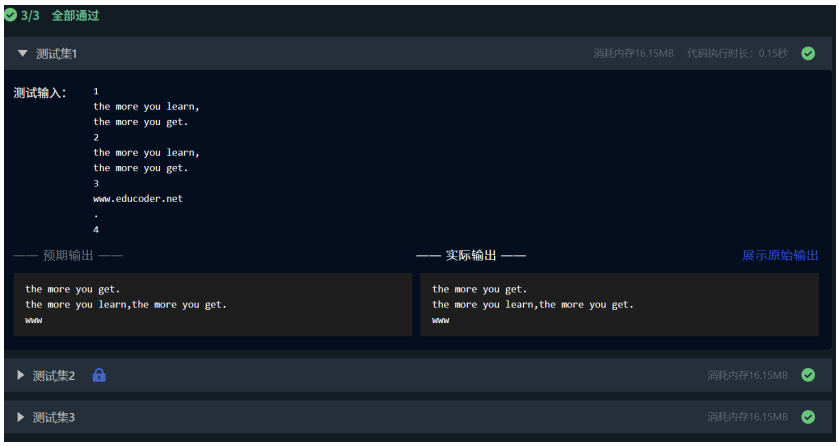


图 1-6　运行截图

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。

```c
#include<stdio.h>
#include<string.h>
int main(){
    int temp;
    char s1[100];    char s2[100];    char *s3;
    char * (*funlist[3])(char *,const char *)={
        strcpy, strcat, strtok
    };
    scanf("%d",&temp);
    getchar();
    while(temp!=4){
        fgets(s1,sizeof(s1),stdin);
        fgets(s2,sizeof(s2),stdin);
        s1[strcspn(s1, "\n")] = '\0';
        s2[strcspn(s2, "\n")] = '\0';
        s3=funlist[temp-1](s1,s2);
        printf("%s\n",s3);
        scanf("%d",&temp);
        getchar();
    }
    return 0;
}
```

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

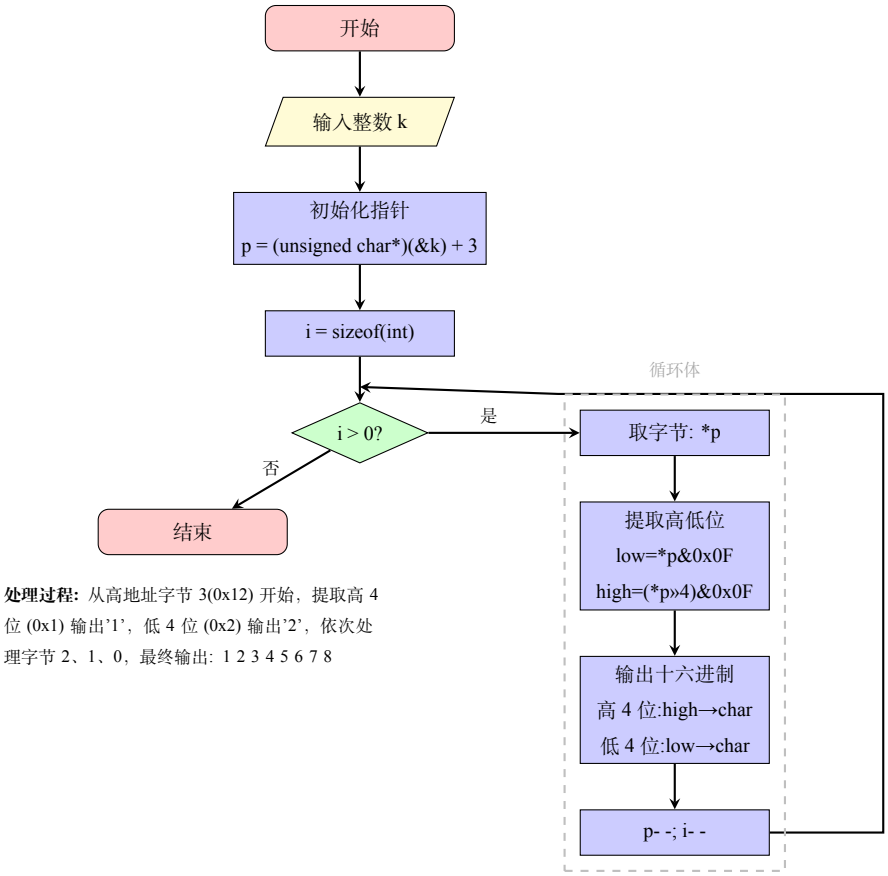


图 1-7 整数按字节分解为十六进制字符的流程图

1.4.3 完整程序代码

```
#include<stdio.h>
int main(){
    int k;
    scanf("%d",&k);
    unsigned char * p=(unsigned char*)(&k)+3;
    for(int i=sizeof(int); i>0; i--){
        unsigned char low = *p & 0x0F;
```

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

Listing 7　实验 6 程序设计 1 完整代码

1.4.4　程序运行示例

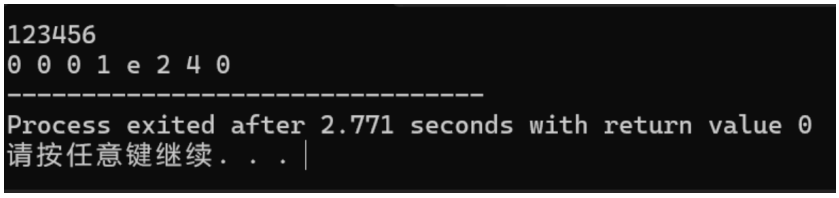


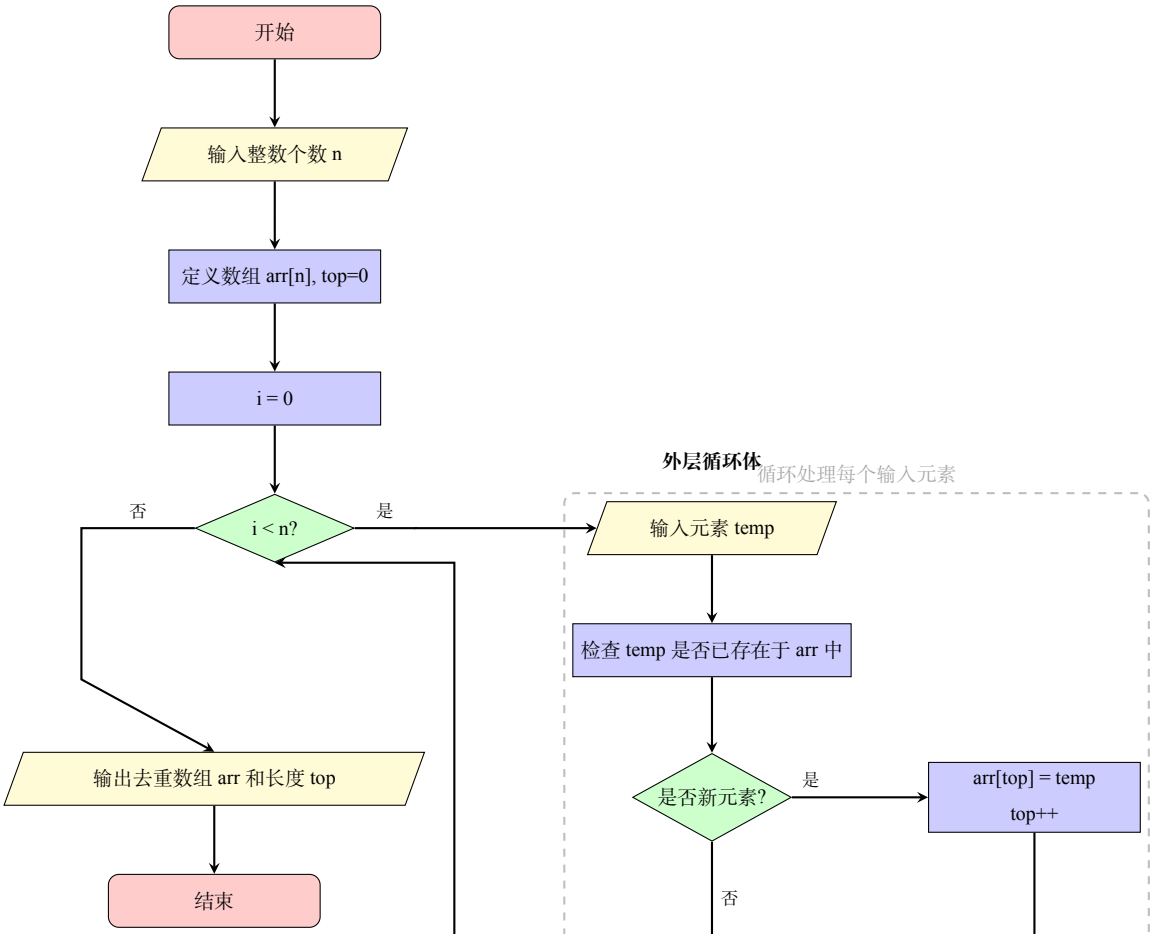
图 1-8　通过指针取出字节

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



算法说明:

- **输入:** n 个整数，可能包含重复元素
- **过程:** 遍历所有输入元素，对于每个元素检查是否已存在于去重数组 arr 中
- **输出:** 输出去重后的数组和其长度
- **时间复杂度:** $O(n^2)$ - 最坏情况下每个元素都要与之前所有元素比较
- **空间复杂度:** O(n) - 需要存储去重结果

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

```c
#include<stdio.h>
int main(){
    int n;
    scanf("%d",&n);
    int arr[n];
    int top=0;
    int temp,flag=0;
    for(int i=0;i<n;i++){
        flag=0;
        scanf("%d",&temp);
        for(int j=0;j<top;j++){
            if(temp==*(arr+j)){
                flag=1;
                break;
            }
        }
        if(!flag){
        arr[top]=temp;
        top+=1;
        }
    }
    for(int i=0;i<top-1;i++){
        printf("%d ",*(arr+i));
    }
    printf("%d",*(arr+top-1));
    printf("\n%d",top);
    return 0;
}
```

Listing 8　实验 6 程序设计 1 完整代码

1.5.4 程序运行示例

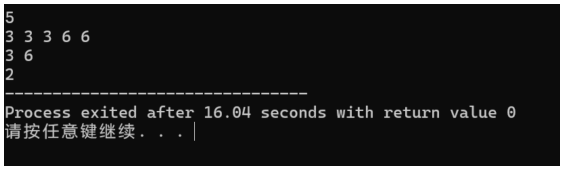


图 1-10　删除重复元素

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

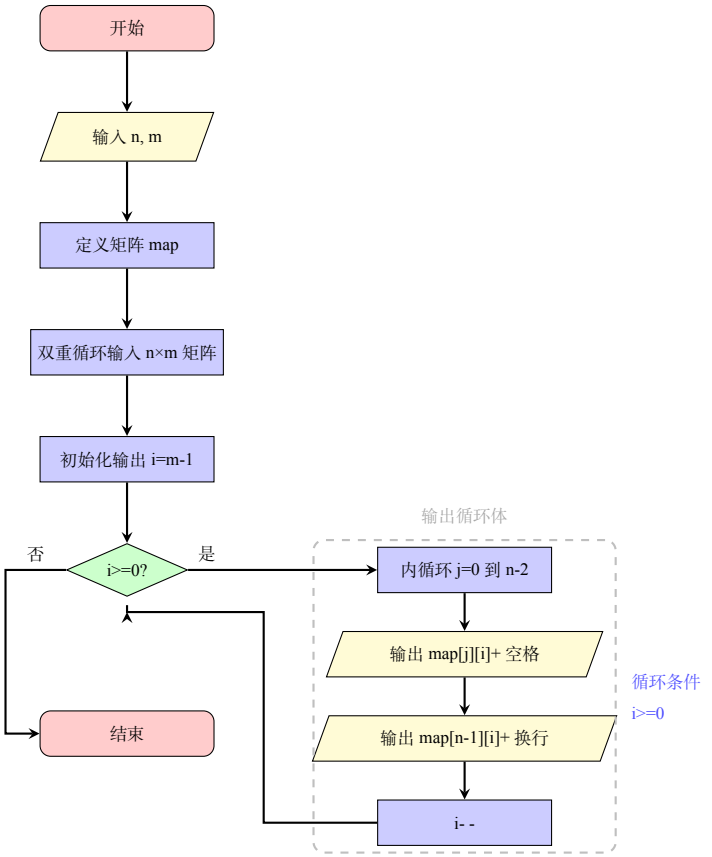


图 1-11 矩阵逆时针旋转 90° 算法流程图（简化版）

1.6.3 完整程序代码

```c
#include<stdio.h>
int main(){
    int n,m;
    scanf("%d%d",&n,&m);
    int map[n][m];
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            scanf("%d",&map[i][j]);
        }
    }
    for(int i=m-1;i>=0;i--){
        for(int j=0;j<n-1;j++){
            printf("%d ",map[j][i]);
        }
        printf("%d\n",map[n-1][i]);
    }
    return 0;
}
```

Listing 9 矩阵逆时针旋转 90° 完整代码

1.6.4 程序运行示例

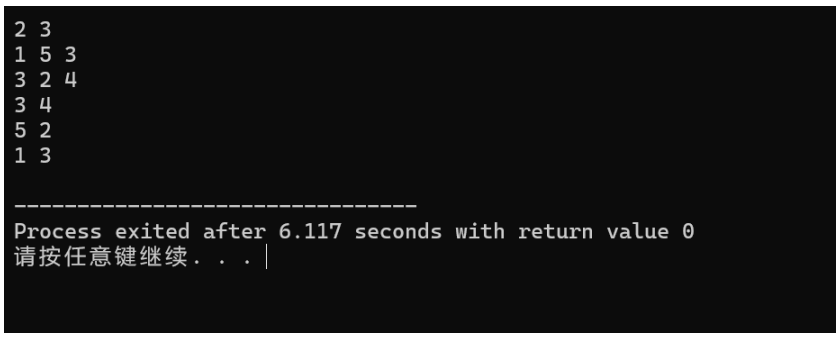


图 1-12 矩阵逆时针旋转

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

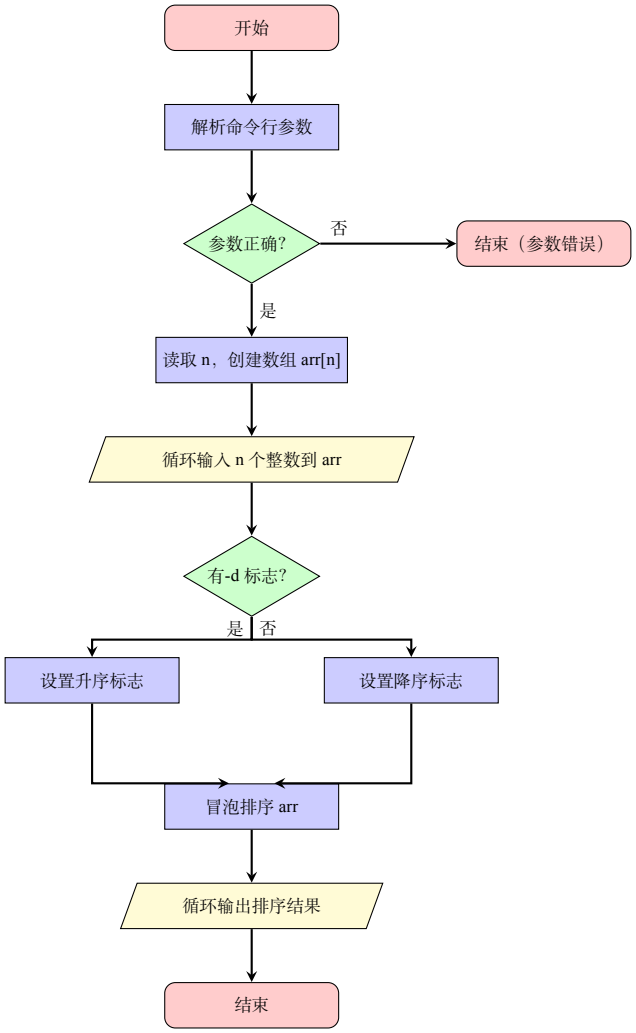


图 1-13 命令行整数排序程序流程图

1.7.3 完整程序代码

```c
#include<stdio.h>
#include<stdlib.h>
void sort(int *arr,int n,int is_desc){
    int temp;
    for(int i=0;i<n;i++){
        for(int j=0;j<n-i-1;j++){
            if((is_desc&&arr[j]<arr[j+1])||(!is_desc&&arr[j]>arr[j+1])){
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}
int main(int argc,char *argv[]){
    if(argc<2){
        printf("Usage:%s N [-d]\n",argv[0]);
        return 1;
    }
    int n=atoi(argv[1]);
    if(n<=0){
        printf("Error:N must be positive integer\n");
        return 1;
    }
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    int is_desc=0;
    if(argc==3&&argv[2][0]=='-'&&argv[2][1]=='d'){
        is_desc=1;
    }
    sort(arr,n,is_desc);
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    return 0;
}
```

Listing 10 命令行整数排序程序完整代码

1.7.4 程序运行示例

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

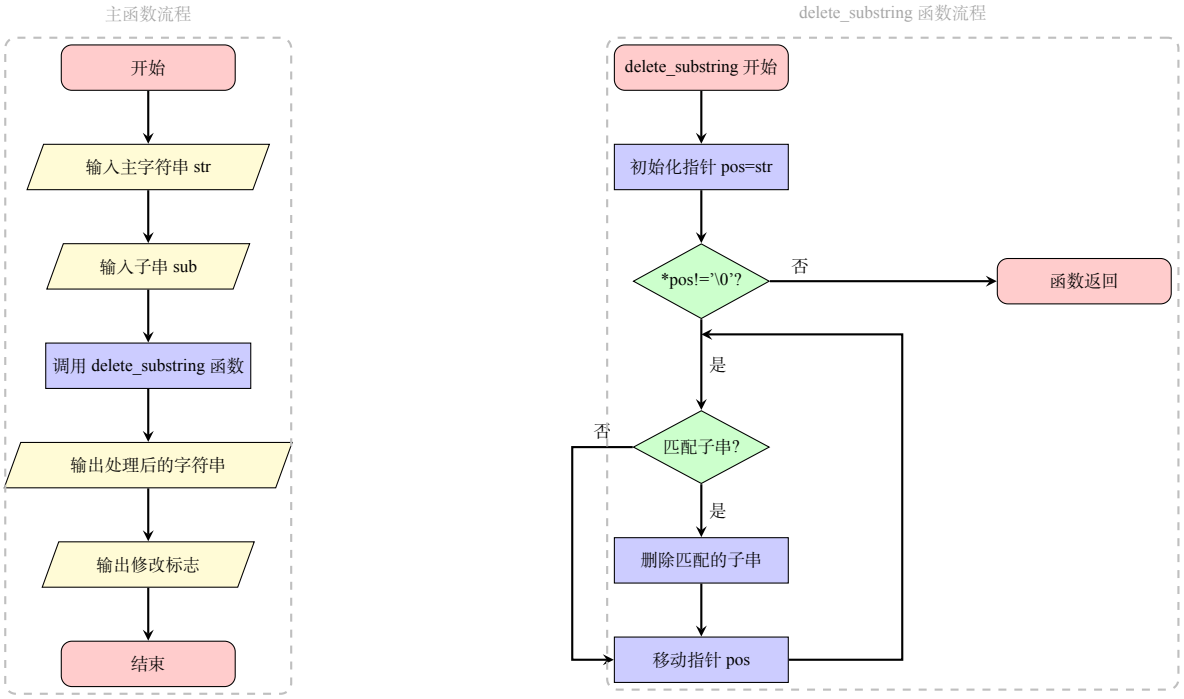


图 1-15 删除子串程序流程图

1.8.3 完整程序代码

```
#include<stdio.h>
#include<string.h>
void delete_substring(char *str,const char *sub){
    char *pos;
    int n=strlen(sub);
    pos=str;
    while(*pos!='\0'){
        int flag=1;
        if((pos-str+n)>str+strlen(str)){
            break;
        }
```

```
        for(int i=0;i<n;i++){
            if(*(pos+i)!=*(sub+i)){
                flag=0;
                break;
            }
        }
        if(flag==1){
            memmove(pos,pos+n,strlen(pos+n)+1);
            pos-=1;
        }
        pos+=1;
    }
}
int main(){
    char str[1000];
    char sub[1000];
    fgets(str,sizeof(str),stdin);
    int n=strlen(str)-1;
    fgets(sub,sizeof(sub),stdin);
    str[strlen(str)-1]='\0';
    delete_substring(str,sub);
    printf("%s\n",str);
    printf("%d",!(strlen(str)==n));
    return 0;
}
```

Listing 11　删除子串程序完整代码

1.8.4　程序运行示例

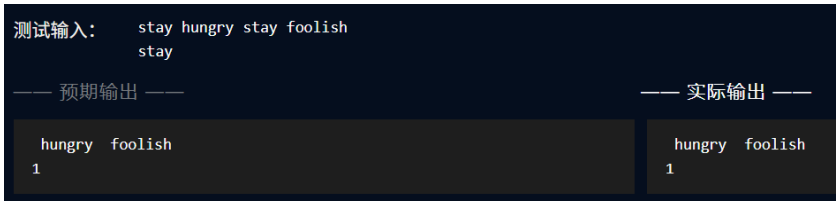


图 1-16　删除字串

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

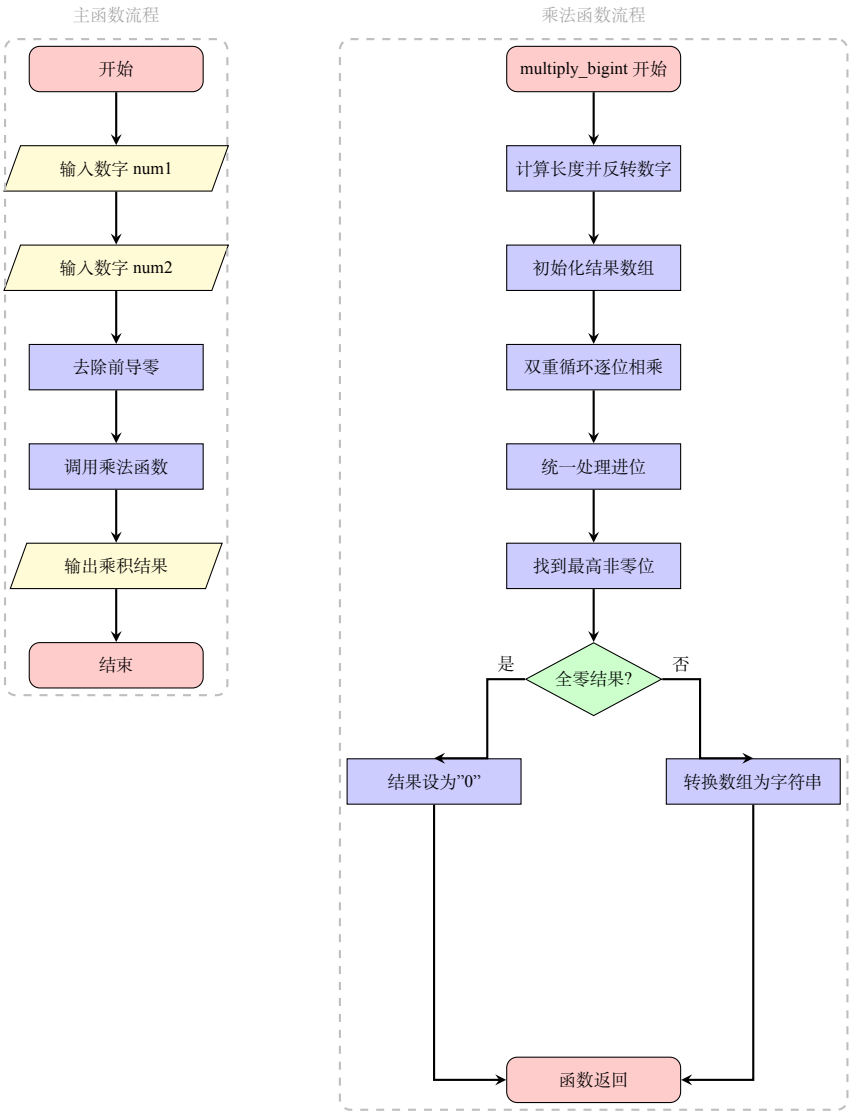


图 1-17 高精度整数乘法程序流程图

1.9.3 完整程序代码

```c
#include<stdio.h>
#include<string.h>
#define MAX_LEN 400
void reverse(char *str){
    int len=strlen(str);
    for(int i=0;i<len/2;i++){
        char temp=str[i];
        str[i]=str[len-1-i];
        str[len-1-i]=temp;
    }
}
void multiply_bigint(char *num1,char *num2,char *result){
    int len1=strlen(num1);
    int len2=strlen(num2);
    int result_len=len1+len2;
    int res[MAX_LEN]={0};
    reverse(num1);
    reverse(num2);
    for(int i=0;i<len1;i++){
        for(int j=0;j<len2;j++){
            res[i+j]+=(num1[i]-'0')*(num2[j]-'0');
            if(res[i+j]>=10){
                res[i+j+1]+=res[i+j]/10;
                res[i+j]%=10;
            }
        }
    }
    for(int i=0;i<result_len;i++){
        if(res[i]>=10){
            res[i+1]+=res[i]/10;
            res[i]%=10;
        }
    }
    int highest=result_len-1;
    while(highest>=0&&res[highest]==0){
        highest--;
    }
    if(highest<0){
        result[0]='0';
        result[1]='\0';
        return;
    }
```

```
    int idx=0;
    for(int i=highest;i>=0;i--){
        result[idx++]=res[i]+'0';
    }
    result[idx]='\0';
}
int main(){
    char num1[MAX_LEN/2+1];
    char num2[MAX_LEN/2+1];
    char result[MAX_LEN+1];
    scanf("%s",num1);
    scanf("%s",num2);
    char *p1=num1;
    char *p2=num2;
    while(*p1=='0'&&*(p1+1)!='\0')p1++;
    while(*p2=='0'&&*(p2+1)!='\0')p2++;
    multiply_bigint(p1,p2,result);
    printf("%s\n",result);
    return 0;
}
```

Listing 12　高精度整数乘法程序完整代码

1.9.4　程序运行示例

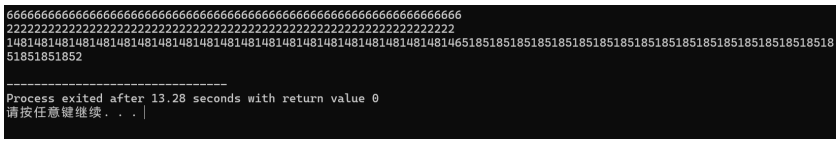


图 1-18　高精度乘法

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

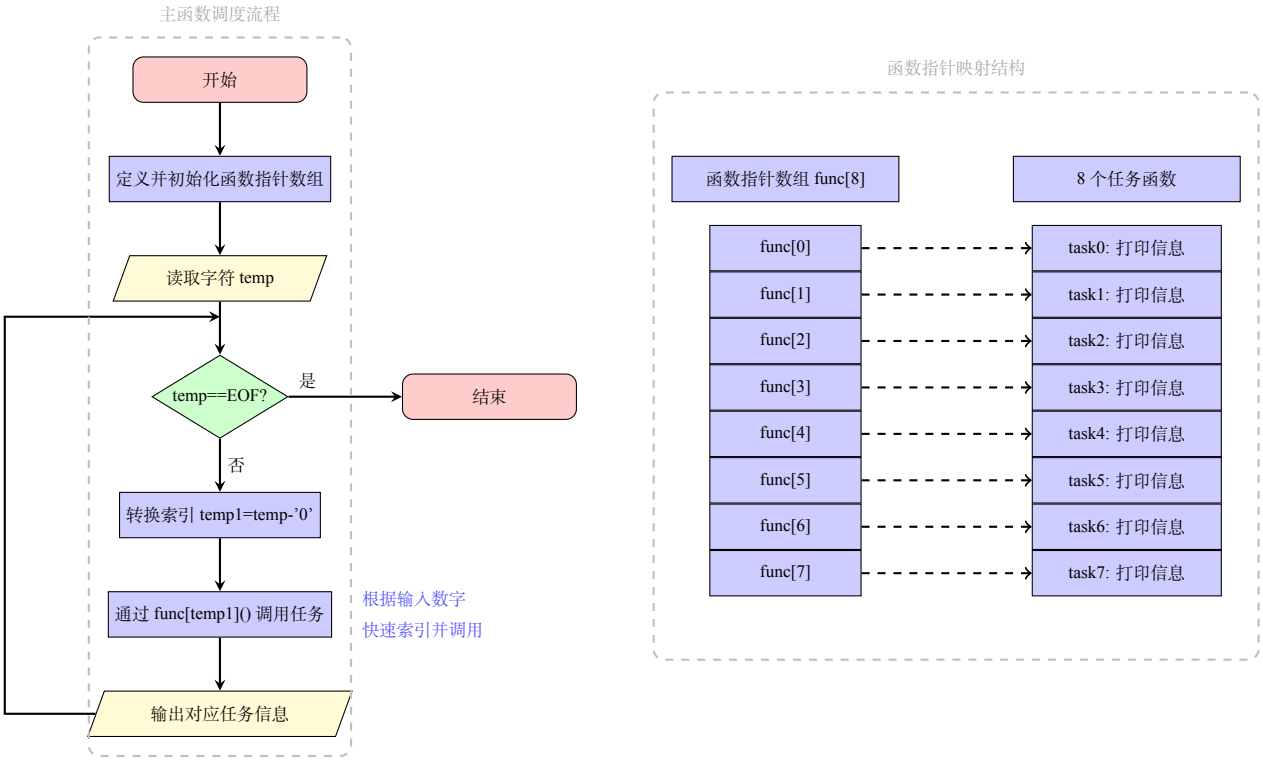


图 1-19 函数调度程序流程图

1.10.3 完整程序代码

```c
#include<stdio.h>
void task0(){
    printf("task0 is called!\n");
}
void task1(){
    printf("task1 is called!\n");
}
void task2(){
    printf("task2 is called!\n");
}
```

```
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

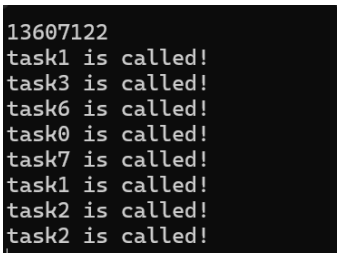


图 1-20　函数调度

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

```
#include<stdio.h>
int main(){
    char u[]="UVWXYZ",v[]="xyz";
    struct T{
        int x;
        char c;
        char *t;
    }
    a[]={{11,'A',u},{100,'B',v}},*p=a;
    int temp;
    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）

```c
#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(void){
    struct s_list *head=NULL,*p;
    int s[]={1,2,3,4,5,6,7,8,0};
    create_list(head,s);
    p=head;
    while (p){
        printf("%d\t",p->data);
        p=p->next;
    }
    printf("\n");
    return 0;
}
void create_list(struct s_list *headp,int *p){
    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

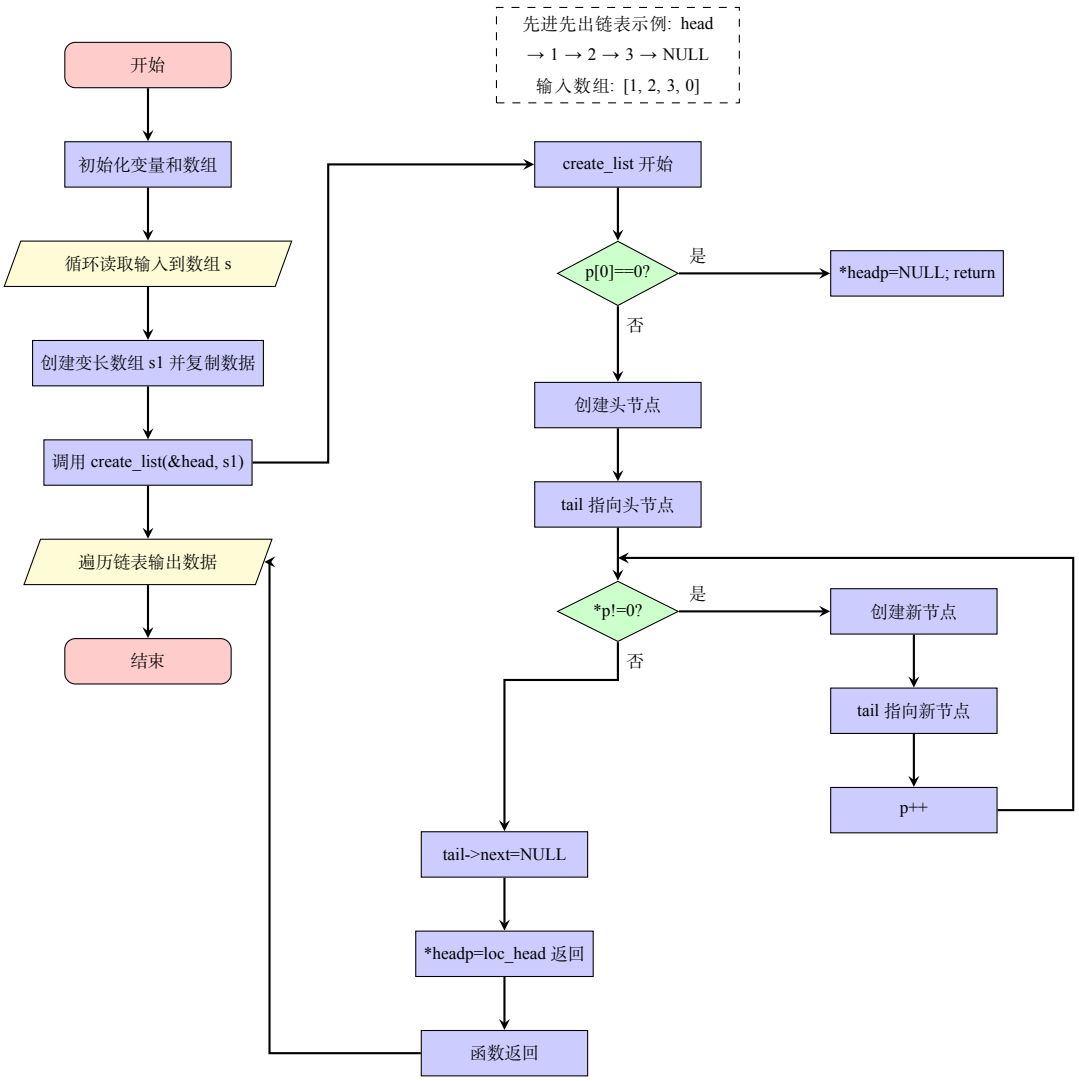


图 2-1 改进后的先进先出链表创建程序流程图

2.2.4 改进方案

针对上述问题，提出以下改进方案：

1. **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。
2. **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。
3. **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

```c
#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(void){
    int s[100];
    struct s_list *head=NULL,*p;
    int top=0;
    int temp;
    while(scanf("%d", &temp) == 1){
        s[top] = temp;
        top += 1;
    }
    int s1[top];
    for(int i=0;i<top;i++){
        s1[i]=s[i];
    }
    create_list(&head,s1);
    p=head;
    while (p){
        printf("%d\t",p->data);
        p=p->next;
    }
    printf("\n");
    return 0;
}
void create_list(struct s_list **headp,int *p){
    struct s_list *loc_head=NULL,*tail;
    if(p[0]==0){
        *headp=NULL;
        return;
    }
    else {
        loc_head=(struct s_list *)malloc(sizeof(struct s_list));
        loc_head->data=*p++;
        tail=loc_head;
        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

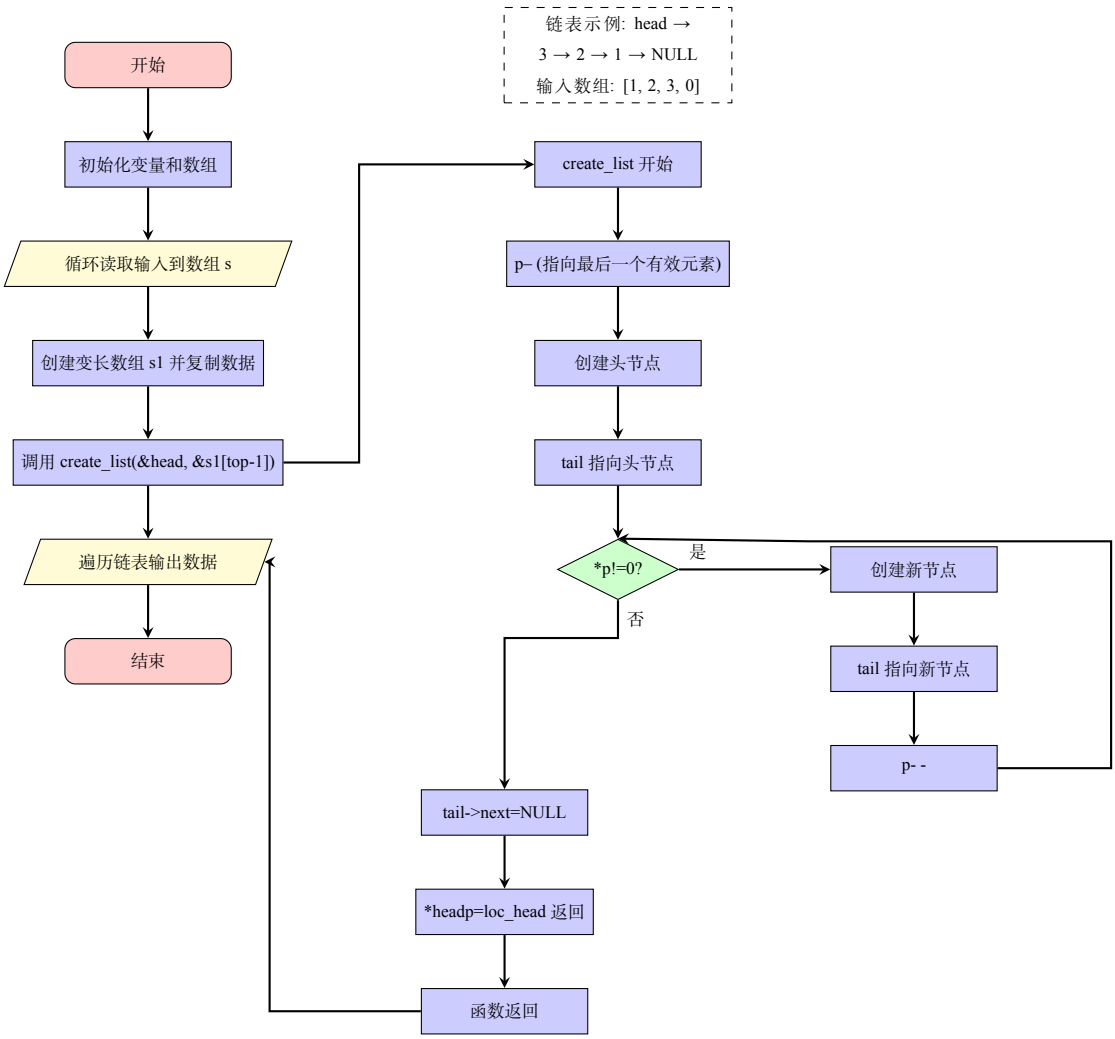


图 2-3 改进后的后进先出链表创建程序流程图

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

```c
#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(void){
    int s[100];
    struct s_list *head=NULL,*p;
    int top=0;
    int temp;
    while(scanf("%d", &temp) == 1){
        s[top] = temp;
        top += 1;
    }
    int s1[top];
    for(int i=0;i<top;i++){
        s1[i]=s[i];
    }
    create_list(&head,&s1[top-1]);
    p=head;
    while (p){
        printf("%d\t",p->data);
        p=p->next;
    }
    printf("\n");
    return 0;
}
void create_list(struct s_list **headp,int *p){
    struct s_list *loc_head=NULL,*tail;
    p--;
    if(1){
        loc_head=(struct s_list *)malloc(sizeof(struct s_list));
        loc_head->data=*p--;
        tail=loc_head;
        while (*p)
        {
```

```
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

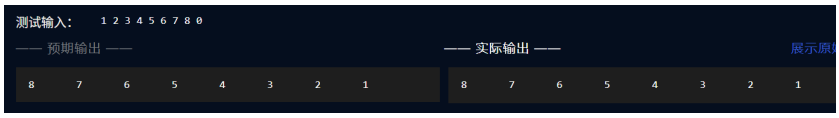


图 2-4　源程序修改替换（二）

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

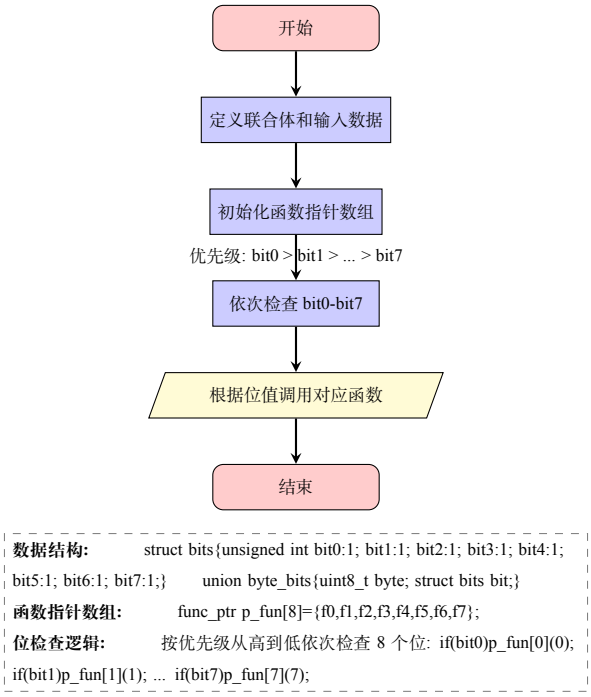


图 2-5 位字段结构与函数指针调度程序流程图

2.4.3 完整程序代码

```c
#include <stdio.h>
#include <stdint.h>
struct bits {
    unsigned int bit0 : 1;
    unsigned int bit1 : 1;
/*此处省略6个定义*/
};
```

```c
union byte_bits {
    uint8_t byte;
    struct bits bit;
};
void f0(int b){printf("the function 0 is called!\n");}
/*此处省略7个声明*/
typedef void (*func_ptr)(int);
int main(){
    union byte_bits data;
    unsigned int input;
    scanf("%u",&input);
    data.byte=(uint8_t)(input&0xFF);
    func_ptr p_fun[8]={f0,f1,f2,f3,f4,f5,f6,f7};
    if(data.bit.bit0)p_fun[0](0);
    if(data.bit.bit1)p_fun[1](1);
/*此处省略6个调用*/
    return 0;
}
```

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

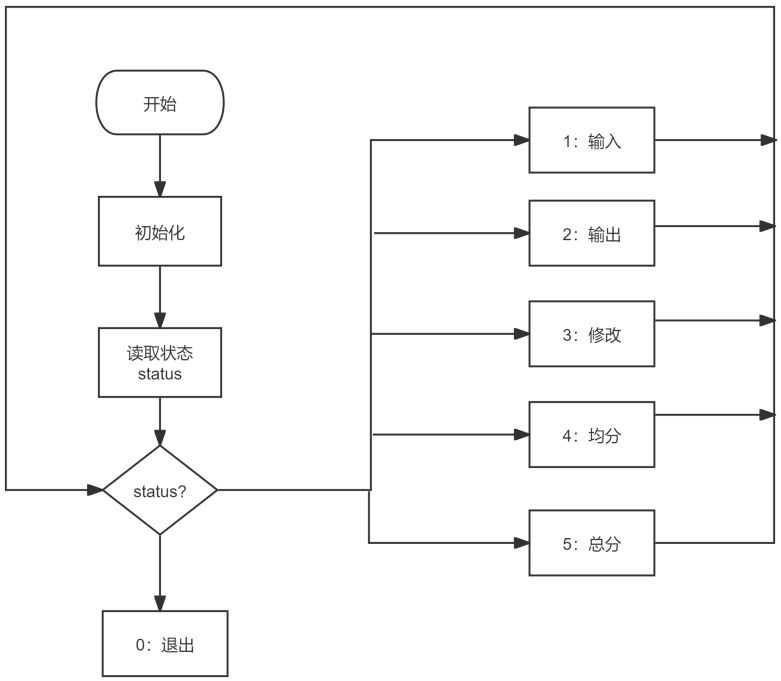


图 2-7 主函数流程图

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。

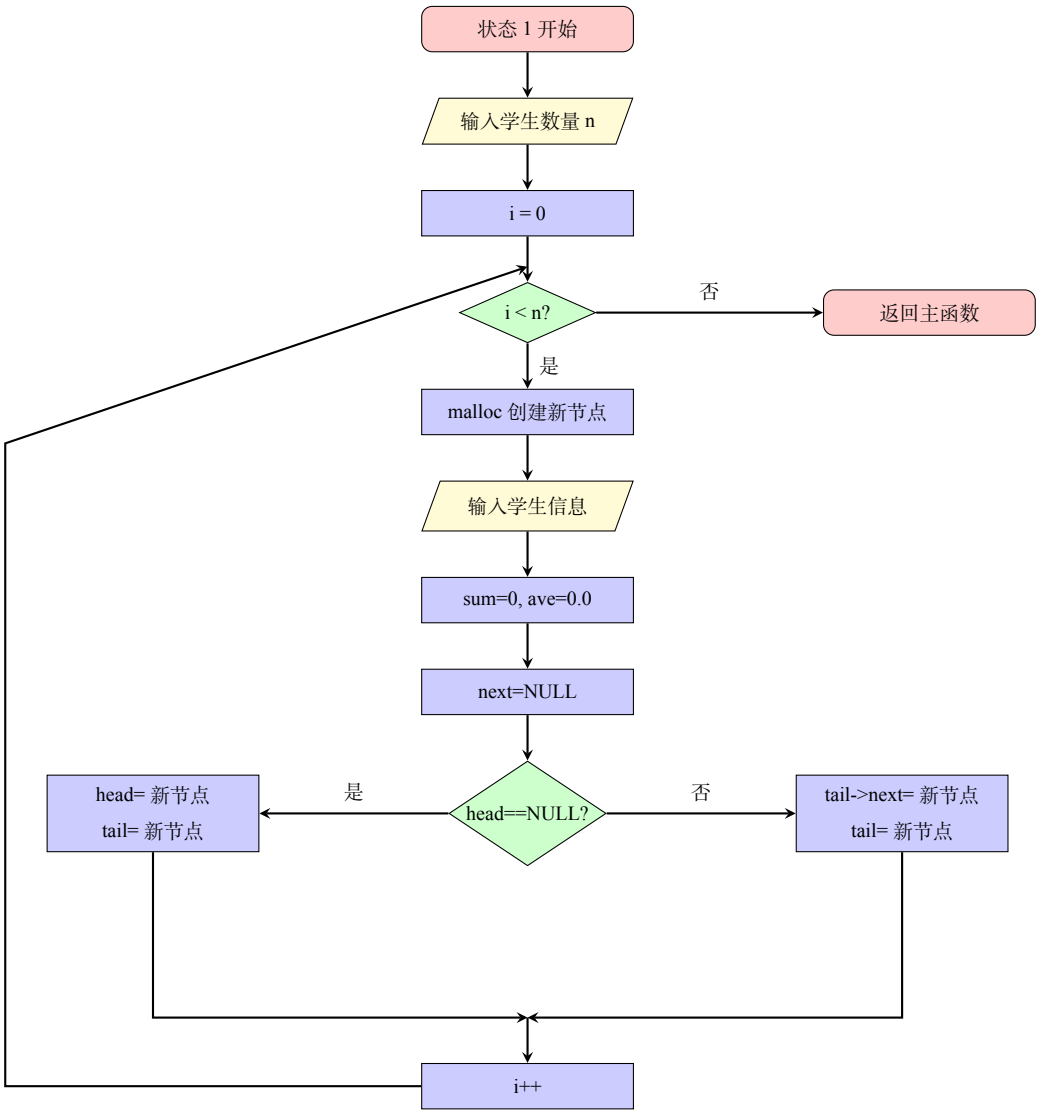


图 2-8　状态 1：输入学生信息流程图（链表实现）

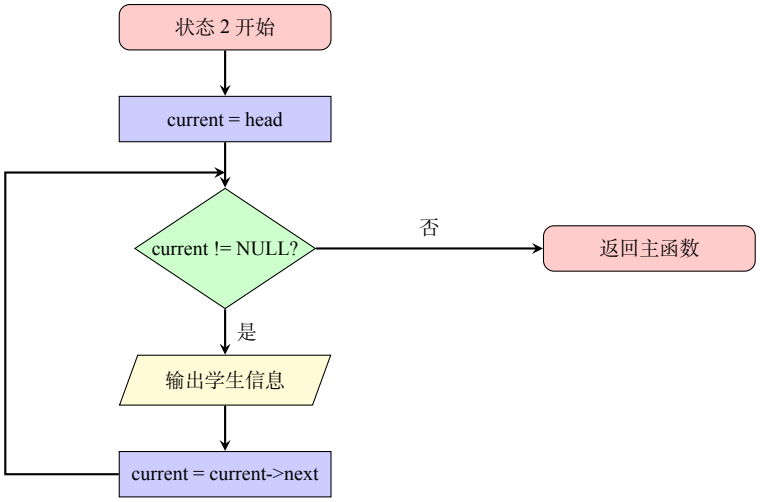


图 2-9　状态 2：输出学生信息流程图（链表实现）

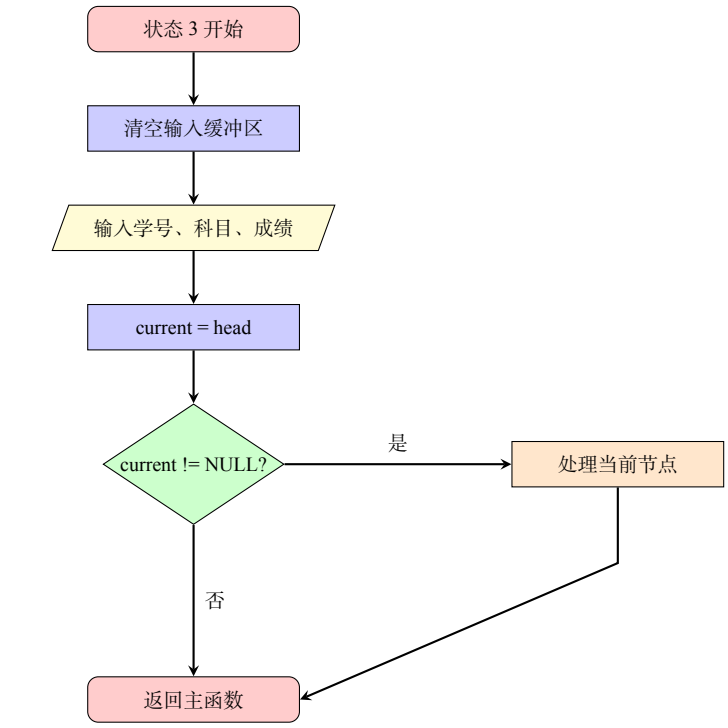


图 2-10　状态 3：修改学生成绩流程图（链表实现）

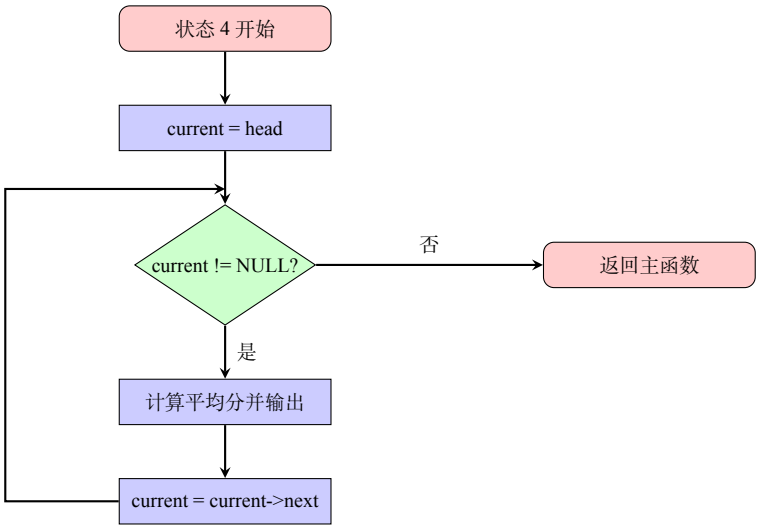


图 2-11　状态 4：计算平均成绩流程图（链表实现）

2.5.3　完整程序代码

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct student {
    char num[11];
    char name[11];
    int score[4];
    int sum;
```

```
    float ave;
    struct student *next;
};
int main() {
    int status;
    struct student *head = NULL;
    struct student *tail = NULL;
    int n;
    while(1) {
        scanf("%d", &status);
        if(status == 0) {
            break;
        }
        if(status == 1) {
            scanf("%d", &n);
            for(int i = 0; i < n; i++) {
                struct student *new_student = (struct
                   student*)malloc(sizeof(struct student));
                scanf("%s %s %d %d %d %d",
                      new_student->num,
                      new_student->name,
                      &new_student->score[0],
                      &new_student->score[1],
                      &new_student->score[2],
                      &new_student->score[3]);
                new_student->sum = 0;
                new_student->ave = 0.0;
                new_student->next = NULL;
                if(head == NULL) {
                    head = new_student;
                    tail = new_student;
                } else {
                    tail->next = new_student;
                    tail = new_student;
                }
            }
        }
        if(status == 2) {
            struct student *current = head;
            while(current != NULL) {
                printf("%s %s %d %d %d %d\n",
                       current->num,
                       current->name,
                       current->score[0],
                       current->score[1],
```

```
                        current->score[2],
                        current->score[3]);
                current = current->next;
            }
        }
        if(status == 3) {
            while(getchar() != '\n');
            char tnum[11];
            int dis;
            int tscore;
            scanf("%s %d %d", tnum, &dis, &tscore);
            struct student *current = head;
            while(current != NULL) {
                if(strcmp(current->num, tnum) == 0) {
                    if(dis >= 1 && dis <= 4) {
                        current->score[dis - 1] = tscore;
                    }
                    break;
                }
                current = current->next;
            }
        }
        if(status == 4) {
            struct student *current = head;
            while(current != NULL) {
                current->sum = 0;
                for(int j = 0; j < 4; j++) {
                    current->sum += current->score[j];
                }
                current->ave = current->sum / 4.0;
                printf("%s %s %.2f\n",
                        current->num,
                        current->name,
                        current->ave);
                current = current->next;
            }
        }
        if(status == 5) {
            struct student *current = head;
            while(current != NULL) {
                if(current->sum == 0) {
                    current->sum = 0;
                    for(int j = 0; j < 4; j++) {
                        current->sum += current->score[j];
                    }
```

```
                    current->ave = current->sum / 4.0;
                }
                printf("%s %s %d %.2f\n",
                        current->num,
                        current->name,
                        current->sum,
                        current->ave);
                current = current->next;
            }
        }
    }
    struct student *current = head;
    while(current != NULL) {
        struct student *temp = current;
        current = current->next;
        free(temp);
    }
    return 0;
}
```

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

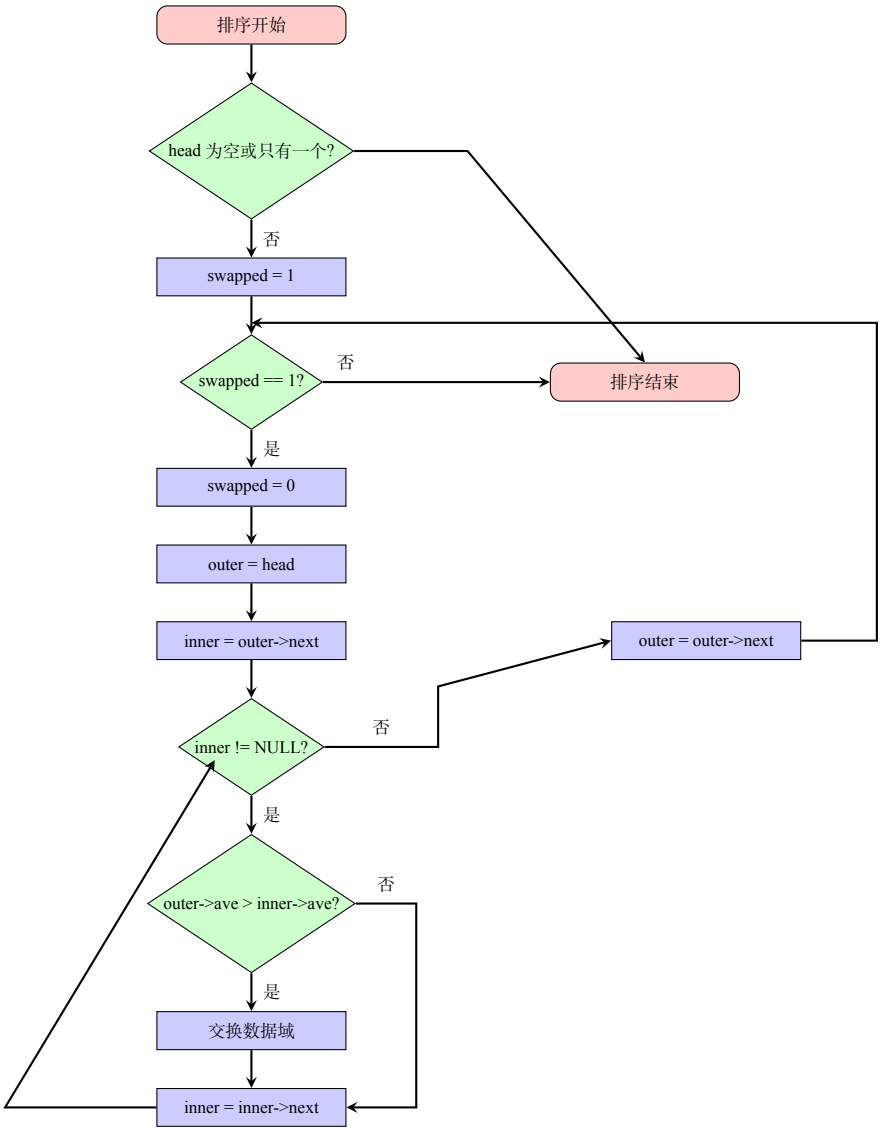


图 2-13 按平均成绩升序排序流程图（冒泡法）

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

```
void swapStudentData(struct student *a, struct student *b) {
    struct student temp;
    strcpy(temp.num, a->num);
    strcpy(temp.name, a->name);
    for(int i = 0; i < 4; i++) temp.score[i] = a->score[i];
    temp.sum = a->sum;
    temp.ave = a->ave;
    strcpy(a->num, b->num);
    strcpy(a->name, b->name);
    for(int i = 0; i < 4; i++) a->score[i] = b->score[i];
    a->sum = b->sum;
    a->ave = b->ave;
    strcpy(b->num, temp.num);
    strcpy(b->name, temp.name);
    for(int i = 0; i < 4; i++) b->score[i] = temp.score[i];
    b->sum = temp.sum;
    b->ave = temp.ave;
}
void sortByAverage(struct student *head) {
    if(head == NULL || head->next == NULL) {
        return;
    }
    struct student *outer, *inner;
    int swapped;
    do {
        swapped = 0;
        outer = head;

        while(outer->next != NULL) {
            inner = outer->next;
            if(outer->ave > inner->ave) {
                swapStudentData(outer, inner);
                swapped = 1;
            }
            outer = outer->next;
        }
    } while(swapped);
}
```

Listing 20 排序算法代码

2.6.4 程序运行示例

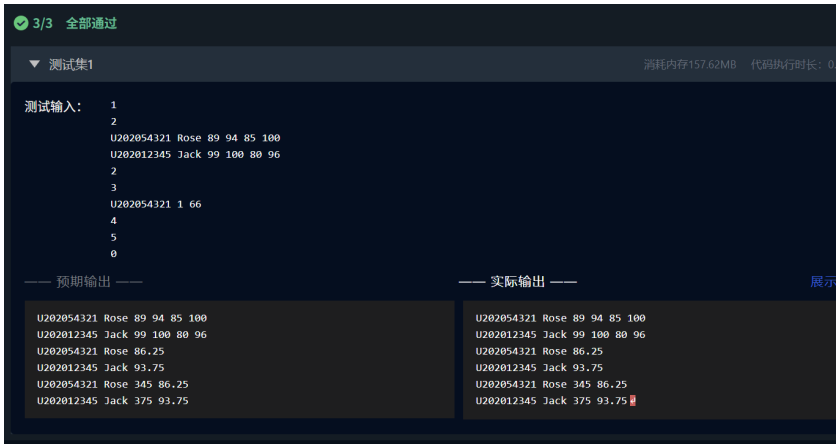


图 2-14 成绩排序（一）

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

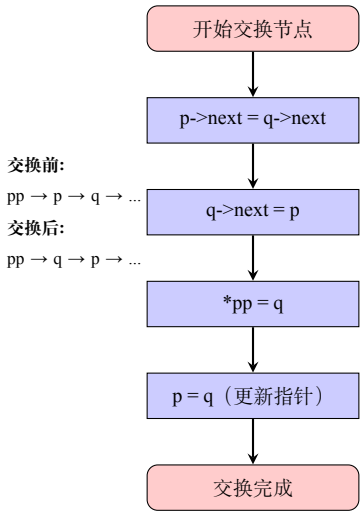


图 2-15 交换相邻节点指针域的详细操作

2.7.3 核心修改代码

```
void swapNodes(struct student **head,struct student *prev,struct
   student *curr,struct student *next){
    if(prev==NULL){
        curr->next=next->next;
        next->next=curr;
        *head=next;
    }else{
        prev->next=next;
        curr->next=next->next;
        next->next=curr;
    }
}
void sortByAverageSelection(struct student **head){
    if(*head==NULL||(*head)->next==NULL)return;
    struct student **pp,*p,**min_pp,*min_p;
    struct student *q;
    for(pp=head;*pp!=NULL;pp=&(*pp)->next){
```

```
        min_pp=pp;
        min_p=*pp;
        for(q=(*pp)->next;q!=NULL;q=q->next){
            if(q->ave<min_p->ave){
                min_pp=&(*pp)->next;
                min_p=q;
            }
        }
        if(min_p!=*pp){
            *min_pp=min_p->next;
            min_p->next=*pp;
            *pp=min_p;
            pp=&min_p;
        }
    }
}
```

Listing 21　排序算法代码

2.7.4　程序运行示例

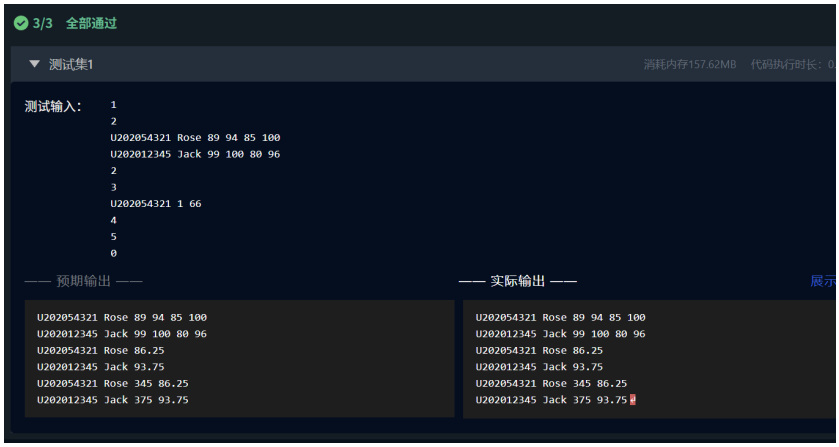


图 2-16　成绩排序（二）

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

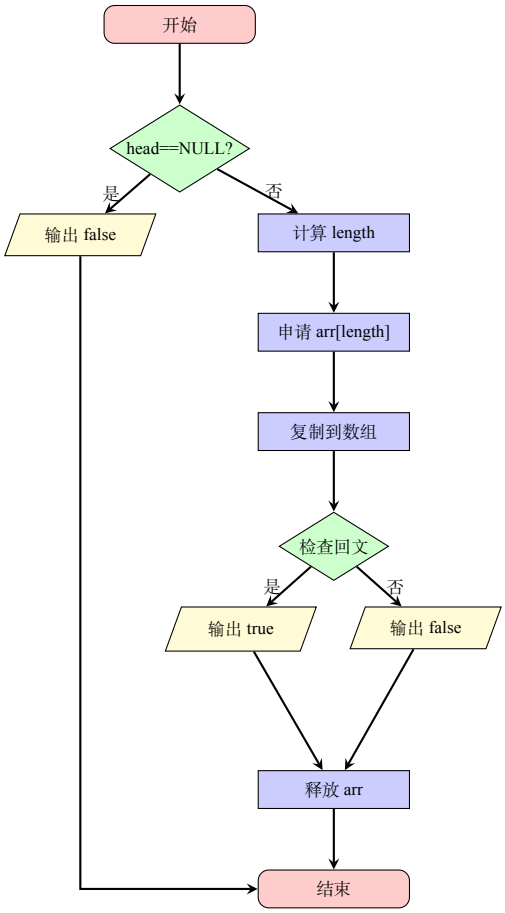


图 2-17 回文字符串判断算法流程图

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++) {
```

```
        C_NODE *new_node = (C_NODE*)malloc(sizeof(C_NODE));
        if(new_node == NULL) {
            return;
        }
        new_node->data = s[i];
        new_node->next = NULL;

        if(*headp == NULL) {
            *headp = new_node;
            tail = new_node;
        } else {
            tail->next = new_node;
            tail = new_node;
        }
    }
}

void judgePalindrome(C_NODE *head) {
    if(head == NULL) {
        printf("false\n");
        return;
    }

    int length = 0;
    C_NODE *current = head;
    while(current != NULL) {
        length++;
        current = current->next;
    }

    char *arr = (char*)malloc(length * sizeof(char));
    if(arr == NULL) {
        printf("false\n");
        return;
    }

    current = head;
    for(int i = 0; i < length; i++) {
        arr[i] = current->data;
        current = current->next;
    }

    int is_palindrome = 1;
    for(int i = 0; i < length / 2; i++) {
        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　回文字符串判断

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

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdbool.h>
typedef struct Node {
    int data;
    struct Node* next;
} Node;
typedef struct {
    Node* top;
} Stack;
void initStack(Stack* stack) {
    stack->top = NULL;
}
bool isEmpty(Stack* stack) {
    return stack->top == NULL;
}
void push(Stack* stack, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = stack->top;
    stack->top = newNode;
}
int pop(Stack* stack) {
    Node* temp = stack->top;
    int value = temp->data;
    stack->top = temp->next;
    free(temp);
    return value;
}
int peek(Stack* stack) {
    return stack->top->data;
```

```
}
void freeStack(Stack* stack) {
    while (!isEmpty(stack)) {
        pop(stack);
    }
}
bool isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
}
int performOperation(int a, int b, char op) {
    switch (op) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return a / b;
        default: return 0;
    }
}
int main() {
    Stack stack;
    initStack(&stack);
    char c;
    int num = 0;
    bool readingNumber = false;
    bool isNegative = false;
    while ((c = getchar()) != EOF) {
        if (isdigit(c)) {
            num = num * 10 + (c - '0');
            readingNumber = true;
        } else if (c == '-') {
            char nextChar = getchar();
            if (isdigit(nextChar)) {
                isNegative = true;
                num = (nextChar - '0');
                readingNumber = true;
                while (isdigit(nextChar = getchar())) {
                    num = num * 10 + (nextChar - '0');
                }
                ungetc(nextChar, stdin);
            } else {
                ungetc(nextChar, stdin);
                if (readingNumber) {
                    if (isNegative) {
                        num = -num;
                        isNegative = false;
```

```
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　逆波兰表达式

2.9.4　程序运行示例

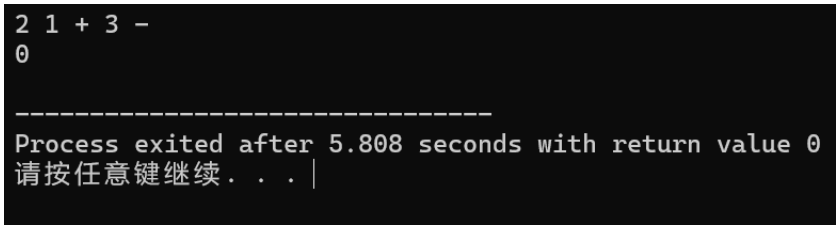


图 2-19　逆波兰表达式

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