華中科技大学

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

学　　号：U202315696

姓　　名：彭冲

指导教师：毛伏兵

报告日期：2023.11.22

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

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

预期输入输出

```
Input a string:
programming↙    (键盘输入)
programming
Input a string again;
language↙       (键盘输入)
language
```

```c
/* 实验 6-1 程序改错与跟踪题源序 */

#include <stdio.h>
char *strcopy(char *, const char *);
int main(void)
{
    char *s1, *s2, *s3;
    printf("Input a string:\n", s2);
    scanf("%s", s2);
    strcopy(s1, s2);
    printf("%s\n", s1);
    printf("Input a string again:\n", s2);
    scanf("%s", s2);
    s3 = strcopy(s1, s2);
    printf("%s\n", s3);
    return 0;
}
/* 将字符串 s 复制给字符串 ,并且返回串的首地址 */
char *strcopy(char *t, const char *s)
{

    while (*t++ = *s++);
    return (t);
}

```

程序跟踪调试与改错

操作：编译。

结果：失败。

```
ex601.c: In function ‘main’:
ex601.c:8:12: warning: too many arguments for format [-Wformat-extra-args]
    8 |     printf("Input a string:\n", s2);
      |            ^~~~~~~~~~~~~~~~~~~
ex601.c:12:12: warning: too many arguments for format [-Wformat-extra-args]
   12 |     printf("Input a string again:\n", s2);
      |            ^~~~~~~~~~~~~~~~~~~~~~~~~~
```

分析：语法错误。第 8 行和 12 行的 printf 函数接收了两个参数，但第一个参数 format 字符串中并没有第二个参数对应的位置。

操作：修改：删除 printf 函数第二个参数。重新编译。

Line 8&12	Code
修改前	`8.  printf("Input a string:\n", s2);`
修改后	`8.  printf("Input a string:\n");`

结果：编译成功

操作：运行程序。

结果：运行失败。

```
~$ ./ex601
Input a string:
programming
Segmentation fault
~$
```

分析：语法错误。区段错误，也称访问权限冲突，一般出现在程序试图访问不允许访问的内存位置。

操作：Debug。

结果：失败，运行第 9 行时显示 Segmentation fault。

分析：语法错误。第 7 行中，s1,s2,s3 均为未初始化的野指针。

操作：修改：将 s1,s2,s3 声明为字符串数组，将 s3 初始化为 NULL。并重新编译运行。

Line 7	Code
修改前	7. char *s1, *s2, *s3;
修改后	7. char s1[1024], s2[1024], *s3 = NULL;

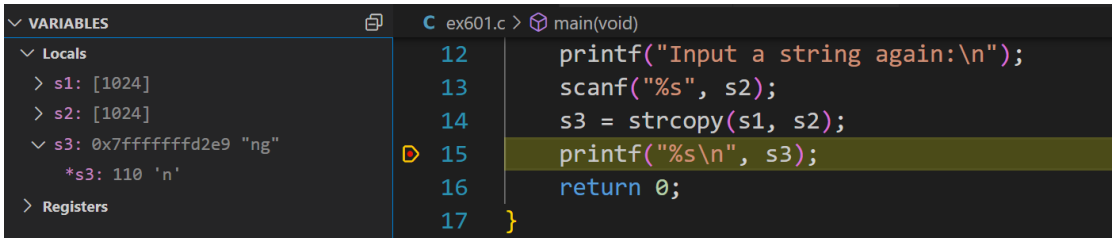
结果：运行成功，但第 2 次输出结果错误。

```
1. ~$ ./ex601
2. Input a string:
3. programming
4. programming
5. Input a string again:
6. language
7. ng
8. ~$
```

分析：逻辑错误。s3 赋值不符合预期。

操作：Debug 跟踪 s3 值。

结果：s3 赋值不符合预期。



分析:逻辑错误。第 2 次输出以 s3 为首字母地址的字符串，而 s3 的值为 strcopy 返回的字符指针 t。当执行第二次 strcopy(s1, s2)时，返回的是 language 字符串的末尾\0 所在地址，即原本的 s1 为 programming\0 经 language\0 覆盖变为 language\0ng\0，此时 s3 为第一个\0 所在地址，故打印 s3 时，打印后面的 ng 直到\0。

操作：修改 strcopy 函数，将返回值改为第一个参数初始值。重新编译运行。

Line 21-23	Code
修改前	21. 22. while (*t++ = *s++); 23. return (t);

修改后	21. char *tmp = t; 22. while (*t++ = *s++); 23. return tmp;

结果：运行成功，输出符合预期。

```
1. ~$ ./ex601
2. Input a string:
3. programming
4. programming
5. Input a string again:
6. language
7. language
8. ~$
```

完毕。

修改后程序为：

```
 1. /* 实验 6-1 程序改错与跟踪题修改后程序 */
 2.
 3. #include <stdio.h>
 4. char *strcopy(char *, const char *);
 5. int main(void)
 6. {
 7.     char s1[1024], s2[1024], *s3 = NULL;
 8.     printf("Input a string:\n");
 9.     scanf("%s", s2);
10.     strcopy(s1, s2);
11.     printf("%s\n", s1);
12.     printf("Input a string again:\n");
13.     scanf("%s", s2);
14.     s3 = strcopy(s1, s2);
15.     printf("%s\n", s3);
16.     return 0;
17. }
18. /* 将字符串 s 复制给字符串 ,并且返回串的首地址 */
19. char *strcopy(char *t, const char *s)
20. {
21.     char *tmp = t;
22.     while (*t++ = *s++)
23.         ;
24.     return tmp;
25. }
26.
```

1.2 程序完善与修改替换

【1.2.1】下面程序中函数 strsort 用于对字符串进行升序排序,在主函数中输入 N 个字符串存入通过 malloc 动态分配的存储空间,然后调用 strsort 对这 N 个串按字典序升序排序。

1. 请在源程序中的下画线处填写合适的代码来完善该程序。

```
1. /* 实验 6 程序完善与修改替换第（1）题源程序：字符串升序排序 */
 2. #include <stdio.h>
 3. #include <______>
 4. #include <string.h>
 5. #define N 4
 6. /* 对指针数组 s 指向的 size 个字符串进行升序排序 */
 7. void strsort(char *s[], int size)
 8. {
 9.     ______temp;
10.     int i, j;
11.     for (i = 0; i < size - 1; i++)
12.         for (j = 0; j < size - i - 1; j++)
13.         {
14.             if (______)
15.             {
16.                 temp = s[j];
17.                 ______;
18.                 s[j+1] = temp;
19.             }
20.         }
21. }
23. int main()
24. {
25.     int i;
26.     char *s[N], t[50];
27.     for (i = 0; i < N; i++)
28.     {
29.         gets(t);
30.         s[i] = (char *)malloc((strlen(t) + 1));
31.         strcpy(______);
32.     }
33.     strsort(______);
34.     for (i = 0; i < N; i++)
35.         puts(s[i]);
36.     return 0;
37. }
```

填充后：

```c
/* 实验 6 程序完善与修改替换第（1）题源程序：字符串升序排序 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 4
/* 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 (s[j][0]>s[j+1][0])
            {
                temp = s[j];
                s[j] = s[j+1];
                s[j+1] = temp;
            }
        }
}

int main()
{
    int i;
    char *s[N], t[50];
    for (i = 0; i < N; i++)
    {
        gets(t);
        //fgets (t, sizeof(t), stdin);
        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;
}

```

运行结果：

```
pttsrd@DESKTOP-6NNOD4C:~/C$ ./ex60201_filled
fghjkl
werty
zxcvbn
tyujk
fghjkl

tyujk

werty

zxcvbn

pttsrd@DESKTOP-6NNOD4C:~/C$
```

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

```c
/* 实验 6 程序完善与修改替换第（1）题源程序：字符串升序排序 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 4
/* 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 (**(s + j) > **(s + j + 1))
            {
                temp = *(s + j);
                *(s + j) = *(s + j + 1);
                *(s + j + 1) = temp;
            }
        }
}

int main()
{
    int i;
```

```
    char *s[N], t[50];
    for (i = 0; i < N; i++)
    {
        gets(t);
        // fgets(t, sizeof(t), stdin);
        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;
}

```

【1.2.2】下面源程序通过函数指针和单选择来调用库丽数实现字符串操作:复制 strcpy、串连接 strcat 或分解 strtok。

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

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

```
/*实验 6-3 程序完善与修改换第(2)题源序:通过函数指针实现字符串操作*/
#include <stdio.h>
#include <string.h>
int main(void)
{
    ______;
    char a[80], b[80], *result;
    int choice;
    while (1)
    {
```

```
11.         do
12.         {
13.             printf("\t\tl copy string.\n");
14.             printf("\t\t2 connect string.\n");
15.             printf("\t\t3 Parse string.\n");
16.             printf("\t\t4 exit.\n");
17.             printf("\t\tinput a number (1-4) please!\n");
18.             scanf("%d", &choice);
19.         } while (choice < 1 || choice > 4);
20.         switch (choice)
21.         {
22.         case 1:
23.             p = strcpy;
24.             break;
25.         case 2:
26.             p = strcat;
27.             break;
28.
29.         case 3:
30.             p = strtok;
31.             break;
32.         case 4:
33.             goto down;
34.         }
35.         getchar();
36.         printf("input the first string please!\n");
37.         ________;
38.         printf("input the second string please!\n");
39.         ________;
40.         result = ____(a, b)
41.         printf("the result is %s\n", result);
42.     }
43. down:
44.     return 0;
45. }
46.
```

填充后：

```
1. /*实验 6-3 程序完善与修改换第(2)题源序:通过函数指针实现字符串操作*/
2. #include <stdio.h>
3. #include <string.h>
4. int main(void)
5. {
6.     char *(*p)(char *s1, const char *s2);
7.     char a[80], b[80], *result;
```

```
 8.     int choice;
 9.     while (1)
10.     {
11.         do
12.         {
13.             printf("\t\tl copy string.\n");
14.             printf("\t\t2 connect string.\n");
15.             printf("\t\t3 Parse string.\n");
16.             printf("\t\t4 exit.\n");
17.             printf("\t\tinput a number (1-4) please!\n");
18.             scanf("%d", &choice);
19.         } while (choice < 1 || choice > 4);
20.         switch (choice)
21.         {
22.         case 1:
23.             p = strcpy;
24.             break;
25.         case 2:
26.             p = strcat;
27.             break;
28.
29.         case 3:
30.             p = strtok;
31.             break;
32.         case 4:
33.             goto down;
34.         }
35.         getchar();
36.         printf("input the first string please!\n");
37.         scanf("%[^\n]%*c", a);
38.         printf("input the second string please!\n");
39.         scanf("%[^\n]%*c", b);
40.         result = p(a, b);
41.         printf("the result is %s\n", result);
42.     }
43. down:
44.     return 0;
45. }
46.
```

运行结果：

```
pttsrd@DESKTOP-6NNOD4C:~/C$ ./ex60221
                1 copy string.
                2 connect string.
                3 Parse string.
                4 exit.
                input a number (1-4) please!
1
input the first string please!
the more you learn,
input the second string please!
genshin impact
the result is genshin impact
                1 copy string.
                2 connect string.
                3 Parse string.
                4 exit.
                input a number (1-4) please!
2
input the first string please!
genshin
input the second string please!
impart
the result is genshinimpart
                1 copy string.
                2 connect string.
                3 Parse string.
                4 exit.
                input a number (1-4) please!
3
input the first string please!
the more you learn, the more you get.
input the second string please!
learn
the result is th
                1 copy string.
                2 connect string.
```

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

```c
/*实验 6-3 程序完善与修改换第(2)题源序:通过函数指针实现字符串操作*/
#include <stdio.h>
#include <string.h>
int main(void)
{
    char *(*p[3])(char *s1, const char *s2) = {strcpy, strcat, strtok};
    char a[80], b[80], *result;
    int choice;
    while (1)
    {
        do
        {
            printf("\t\tl copy string.\n");
            printf("\t\t2 connect string.\n");
            printf("\t\t3 Parse string.\n");
            printf("\t\t4 exit.\n");
            printf("\t\tinput a number (1-4) please!\n");
```

```
            scanf("%d", &choice);
        } while (choice < 1 || choice > 4);

        getchar();
        printf("input the first string please!\n");
        scanf("%[^\n]%*c", a);
        printf("input the second string please!\n");
        scanf("%[^\n]%*c", b);
        result = p[choice-1](a, b);
        printf("the result is %s\n", result);
    }
down:
    return 0;
}

```

运行结果；

```
pttsrd@DESKTOP-6NNOD4C:~/C$ ./ex60222
                1 copy string.
                2 connect string.
                3 Parse string.
                4 exit.
                input a number (1-4) please!
1
input the first string please!
asdfggh
input the second string please!
qwer
the result is qwer
                1 copy string.
                2 connect string.
                3 Parse string.
                4 exit.
                input a number (1-4) please!
2
input the first string please!
asdfgh
input the second string please!
qwer
the result is asdfghqwer
                1 copy string.
                2 connect string.
                3 Parse string.
                4 exit.
                input a number (1-4) please!
3
input the first string please!
asdfghjkl
input the second string please!
h
the result is asdfg
                1 copy string.
                2 connect string.
```

1.3 程序设计

1. 指针取出每个字节

解题思路：

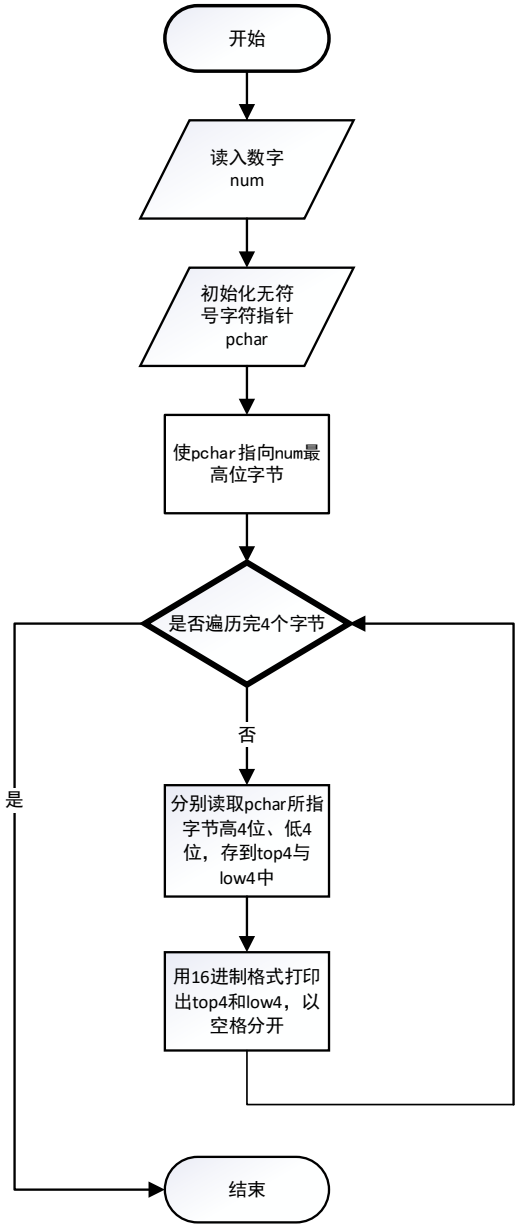


图 6-1 指针取出每个字节流程图

程序代码：

```
1. #include <stdio.h>
2.
3. int main(void)
```

```c
{
    int num;
    scanf("%d", &num); // 输入数字到 num

    unsigned char *pchar;             // 定义无符号字符指针 pchar 用于取出单字节
    pchar = (unsigned char *)&num + 3; // 将输入数据 num 的最高位字节地址赋值给 pchar

    for (int i = 0; i < 4; i++) // 循环 4 次取字节
    {
        char top4 = *pchar >> 4;      // 取字节的高 4 位，由于定义为无符号字符类型，因此最高位 bit 不会自动补 1
        char low4 = *pchar & 0b1111;  // 取字节的低 4 位
        printf("%x %x ", top4, low4); // 用%x 格式输出 16 进制数
        pchar--;                      // 指针低移动一个字节
    }

    putchar('\n');

    return 0;
}

```

运行程序：

```
~$ ./ex60301
123456
0 0 0 1 e 2 4 0
~$
```

2. 删除重复元素

解题思路：

1. 从输入中读取序列的长度和具体的元素。
2. 使用动态内存分配来存储输入序列和输出序列。
3. 初始化当前元素为第一个输入的元素，并将其放入输出序列中。
4. 遍历输入序列，如果当前输入元素小于或等于当前元素，则跳过当前元素；如果当前输入元素大于当前元素，则将其放入输出序列并更新当前元素。
5. 输出去重后的序列和序列的长度。
6. 释放动态分配的内存。

程序代码：

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

int main(int argc, char const *argv[])
{
    int num;
    scanf("%d", &num); // 输入序列的长度

    int *arr_in = malloc(num * sizeof(int)); // 分配内存用于存储输入序列
    for (int i = 0; i < num; i++)
    {
        scanf("%d", arr_in + i); // 逐个输入序列中的元素
    }

    int *arr_out = malloc(num * sizeof(int)); // 分配内存用于存储输出序列

    int cur = *arr_in; // 初始化当前元素为第一个输入的元素
    *arr_out = cur; // 将第一个输入的元素放入输出序列
    int cnt = 1; // 初始化计数器为 1

    for (int i = 1; i < num; i++)
    {
        if (*(arr_in+i) <= cur) // 如果当前输入元素小于或等于当前元素
        {
            continue; // 跳过当前元素，继续下一个
```

```
        }
        else
        {
            *(arr_out+cnt) = cur = *(arr_in+i); // 当前输入元素大于当前元素，将其放入输出序列
            cnt++; // 计数器加一
        }
    }

    // 输出去重后的序列
    printf("%d", *arr_out); // 输出第一个元素
    for (int i = 1; i < cnt; i++)
    {
        printf(" %d", *(arr_out+i)); // 逐个输出序列中的元素
    }
    printf("\n%d", cnt); // 输出去重后序列的长度

    free(arr_in); // 释放输入序列内存
    free(arr_out); // 释放输出序列内存
    return 0;
}

```

运行程序：

```
~$ ./ex60302
6
2 4 4 6 6 7
2 4 6 7
4
~$
```

3. 旋转图像

1. 解题思路：
2. 从输入中读取图像矩阵的行数和列数。
3. 使用动态内存分配创建矩阵。
4. 遍历矩阵，按照逆时针旋转的顺序输出矩阵元素。
5. 外层循环遍历列，从最右侧列开始。
6. 内层循环遍历行，从第一行到最后一行。
7. 释放动态分配的内存。

程序代码：

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

int main(int argc, char const *argv[])
{
    int row, column;
    scanf("%d %d", &row, &column); // 输入图像矩阵的行数和列数

    // 动态分配二维数组来存储图像矩阵
    int **matrix = (int **)malloc(row * sizeof(int *));
    for (int i = 0; i < row; i++)
    {
        matrix[i] = (int *)malloc(column * sizeof(int));
    }

    // 循环读取输入的图像矩阵
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < column; j++)
        {
            scanf("%d", &matrix[i][j]);
        }
    }

    // 逆时针旋转 90°后的矩阵输出
    for (int j = 0; j < column; j++)
    {
        // 按行循环输出
        printf("%d", matrix[0][column - j - 1]);
        for (int i = 1; i < row; i++)
```

```
31.         {
32.             printf(" %d", matrix[i][column - j - 1]);
33.         }
34.         putchar('\n'); // 换行
35.     }
36.
37.     // 释放动态分配的内存
38.     for (int i = 0; i < row; i++)
39.     {
40.         free(matrix[i]);
41.     }
42.     free(matrix);
43.     return 0;
44. }
45.
```

运行程序：

```
~$ ./ex60303
2 3
1 5 3
3 2 4
3 4
5 2
1 3
~$
```

4. 命令行实现对 N 个整数排序

解题思路：

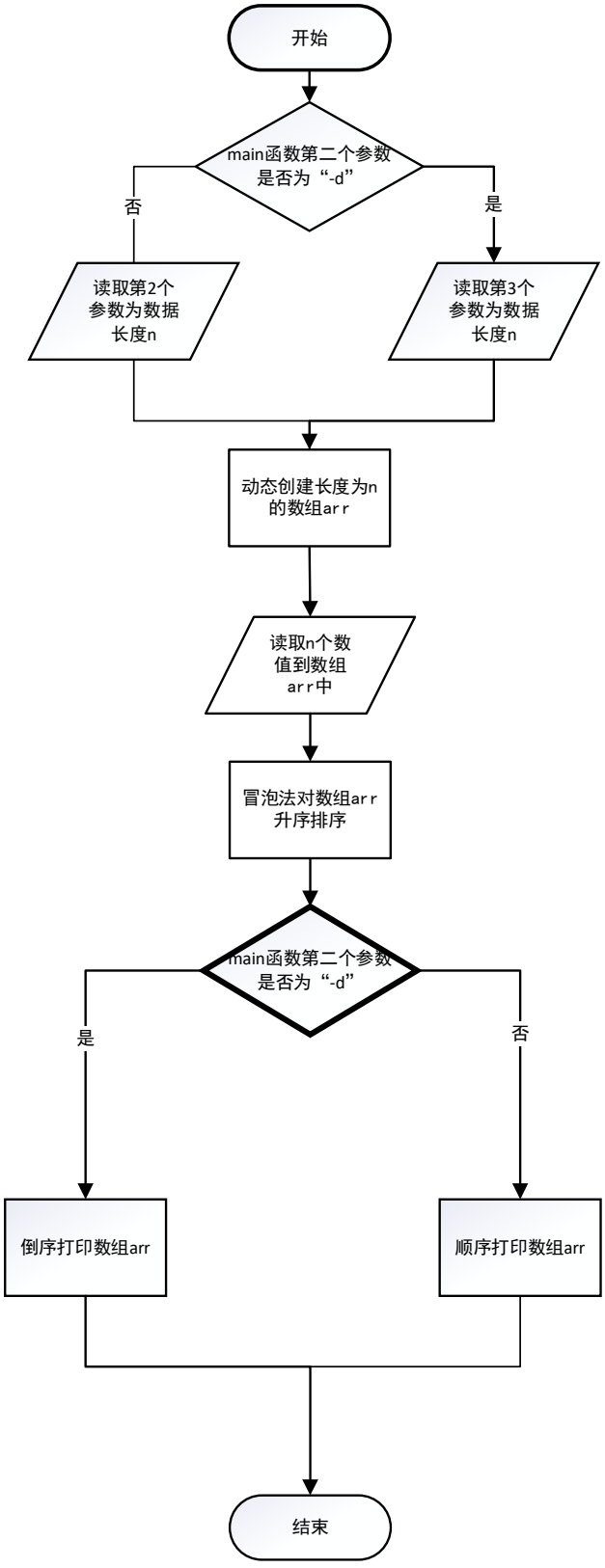


图 6-2 命令行实现对 N 个整数排序的流程图

程序代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 将字符串转换为整数
int Str2Int(const char *s)
{
    int len = strlen(s);
    int o = 0;
    for (int i = 0; i < len; i++)
    {
        o *= 10;
        o += s[i] - '0';
    }
    return o;
}

// 对整数数组进行冒泡排序
void sort(int table[], int cnt)
{
    for (int i = 0; i < cnt - 1; i++)
    {
        for (int j = 0; j < cnt - 1 - i; j++)
        {
            if (table[j] > table[j + 1])
            {
                int tmp = table[j];
                table[j] = table[j + 1];
                table[j + 1] = tmp;
            }
        }
    }
}

int main(int argc, char const *argv[])
{
    int cnt = Str2Int(argv[1]); // 命令行参数第一个参数为待排序的整数个数
    int *table = (int *)malloc(cnt * sizeof(int)); // 动态分配数组用于存储待排序的整数

    // 循环读入待排序的整数
    for (int i = 0; i < cnt; i++)
```

```
42.     {
43.         scanf("%d", &table[i]);
44.     }
45.
46.     sort(table, cnt); // 对待排序的整数进行排序
47.
48.     // 根据命令行参数输出排序后的结果
49.     if (argc == 2)
50.     {
51.         for (int i = 0; i < cnt; i++)
52.         {
53.             printf("%d ", table[i]);
54.         }
55.     }
56.     else if (argc == 3 && !strcmp(argv[2], "-d"))
57.     {
58.         for (int i = 0; i < cnt; i++)
59.         {
60.             printf("%d ", table[cnt - 1 - i]);
61.         }
62.     }
63.
64.     free(table); // 释放动态分配的数组内存
65.     return 0;
66. }
67.
```

运行程序：

```
~$ ./sort  12 -d
4 3 8 5 1 3 6 7 8 9 10 12
12 10 9 8 8 7 6 5 4 3 3 1
~$
```

5. 删除子串

解题思路：

1. 通过 scanf 函数从输入中读取两个字符串，分别存储在 sentence 和 sep 中，其中 sentence 是待处理的字符串，sep 是要删除的子串。
2. 使用 strstr 函数在 sentence 中查找第一次出现 sep 的位置，如果找到了，则将子串之前的部分复制到临时数组 tmp 中，并输出该部分。接着更新起始位置 start_prev，继续在剩余部分中查找子串。
3. 继续使用 strstr 函数查找剩余部分中的子串，找到后同样进行处理，直到再也找不到子串为止。
4. 输出最后一个子串之后的部分，并输出子串出现的次数。

程序代码：

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[])
{
    char sentence[1024], sep[1024];
    scanf("%[^\n]\n%[^\n]", sentence, sep); // 从输入中读取两个字符串，分别存储在 sentence 和 sep 中

    int cnt = 0; // 用于记录子串出现的次数
    char *start_prev;
    char *start = strstr(sentence, sep); // 在 sentence 中查找第一次出现 sep 的位置
    char tmp[1024]; // 临时存储字符串的数组

    if (start != NULL)
    {
        cnt++; // 子串出现次数加 1
        strncpy(tmp, sentence, start - sentence); // 复制子串之前的部分到 tmp 中
        tmp[start - sentence] = '\0'; // 添加字符串结束符
        printf("%s", tmp); // 输出删除子串后的结果
        start_prev = start; // 更新起始位置
        start = strstr(start + strlen(sep), sep); // 继续在剩余部分中查找子串
    }
    else
    {
        printf("%s\n0", sentence); // 如果没有找到子串，直接输出原字符串并返回 0
        return 0;
    }
```

```
29.
30.     while (start != NULL)
31.     {
32.         cnt++; // 子串出现次数加 1
33.          strncpy(tmp, start_prev + strlen(sep), start - start_prev - strlen(sep)); //
复制两个子串之间的部分到 tmp 中
34.         tmp[start - start_prev - strlen(sep)] = '\0'; // 添加字符串结束符
35.         printf("%s", tmp); // 输出删除子串后的结果
36.         start_prev = start; // 更新起始位置
37.         start = strstr(start + strlen(sep), sep); // 继续在剩余部分中查找子串
38.     }
39.     printf("%s", start_prev + strlen(sep)); // 输出最后一个子串之后的部分
40.     printf("\n%d", cnt); // 输出子串出现的次数
41.
42.     return 0;
43. }
44.
```

运行程序：

```
~$ ./ex60305
stay hungry stay foolish
stay
 hungry  foolish
1
~$
```

6. 非负整数积

解题思路：

模拟乘法手算

1. 从输入中读取两个字符串，分别存储在字符数组 s1 和 s2 中；
2. 将字符串中的字符转换为数字，并存储到整型数组 facotr1 和 facotr2 中；
3. 模拟乘法手算，定义二维数组 grid，用于存储乘法结果的每一位；
4. 计算乘法结果的每一位，并存储到 grid 中；
5. 定义数组 result，用于存储最终的乘积结果；
6. 将 grid 中的每一位相加得到 result 的每一位，并处理进位；
7. 输出最终的乘积结果。

程序代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define Length 201 // 定义常量 Length 为最大长度 201

int main(int argc, char const *argv[])
{
    char s1[Length], s2[Length];
    scanf("%s\n%s", s1, s2); // 从输入中读取两个字符串，分别存储在 s1 和 s2 中

    int len1 = strlen(s1); // 获取 s1 的长度
    int len2 = strlen(s2); // 获取 s2 的长度

    int facotr1[Length] = {0}, facotr2[Length] = {0}; // 定义数组 facotr1 和 facotr2，用于存储将字符转换为数字后的结果

    for (int i = 0; i < len1; i++)
    {
        facotr1[i] = s1[len1 - 1 - i] - '0'; // 将 s1 中的字符转换为数字并存储到 facotr1 中
    }

    for (int i = 0; i < len2; i++)
    {
        facotr2[i] = s2[len2 - 1 - i] - '0'; // 将 s2 中的字符转换为数字并存储到 facotr2 中
    }

    int grid[Length][Length * 2] = {{0}}; // 定义二维数组 grid，用于存储乘法结果的每一位

    for (int i = 0; i < len1; i++)
    {
```

```c
        for (int j = 0; j < len2; j++)
        {
            grid[i][j + i] += facotr1[i] * facotr2[j] % 10; // 计算乘法结果的个位数并存储到 grid 中
            grid[i][j + i + 1] += (facotr1[i] * facotr2[j]) / 10; // 计算乘法结果的十位数并存储到 grid 中
        }
    }

    int result[Length * 2] = {0}; // 定义数组 result，用于存储最终的乘积结果

    for (int i = 0; i < Length * 2 - 1; i++)
    {
        for (int j = 0; j < Length; j++)
        {
            result[i] += grid[j][i]; // 将 grid 中的每一位相加得到 result 的每一位
        }
        result[i + 1] += result[i] / 10; // 处理进位
        result[i] %= 10; // 取余得到个位数
    }

    int vacant = Length * 2 - 1; // 空位指针
    while (result[vacant] == 0)
    {
        vacant--; // 找到最高位非零数字的位置
    }

    for (int i = vacant; i >= 0; i--)
    {
        printf("%d", result[i]); // 从最高位开始输出结果
    }

    return 0;
}

```

运行程序：

```
~$ ./ex60306
6666666666666666666666666666666666666666666666666666666666666666
2222222222222222222222222222222222222222222222222222222222222222
1481481481481481481481481481481481481481481481481481481481465185
18518518518518518518518518518518518518518518518518518518512
~$
```

7. 函数调度

解题思路：

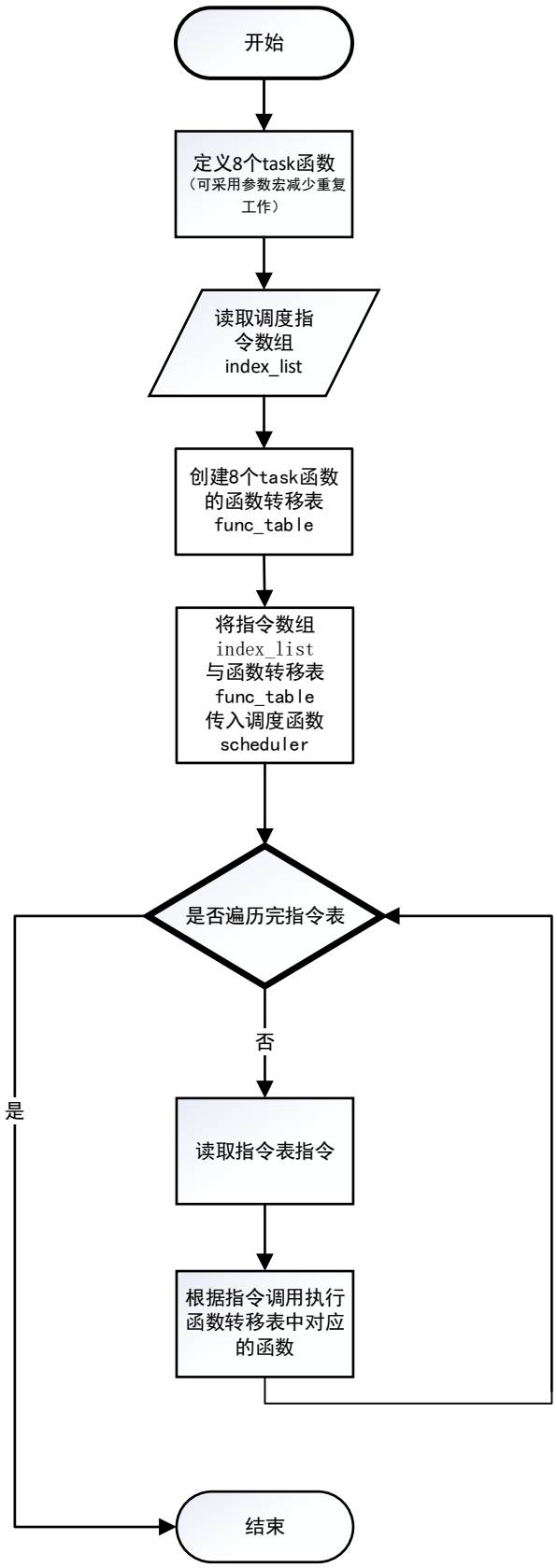


图 6-3 函数调度的流程图

程序代码：

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

#define TSKID(id) task##id // 定义宏，用于生成函数名
#define TSK(id) void TSKID(id)() { printf("task%s is called!\n", #id); } // 定义函数，打印任务被调用的信息

TSK(0) // 定义任务 0
TSK(1) // 定义任务 1
TSK(2) // 定义任务 2
TSK(3) // 定义任务 3
TSK(4) // 定义任务 4
TSK(5) // 定义任务 5
TSK(6) // 定义任务 6
TSK(7) // 定义任务 7

void execute(void (*func_list[])(), int len)
{
    for (int i = 0; i < len; i++)
    {
        func_list[i](); // 调用每个任务函数
    }
}

void scheduler(int *index_list, void (**func_table)())
{
    char sequence[128];
    scanf("%s", sequence); // 从输入中读取任务序列
    int len = strlen(sequence); // 获取任务序列的长度
    index_list = (int *)malloc(len * sizeof(int)); // 分配存储任务序列索引的内存空间
    for (int i = 0; i < len; i++)
    {
        index_list[i] = sequence[i] - '0'; // 将字符转换为整型存储到任务索引列表中
    }
    void (**func_list)() = (void (**)())malloc(len * sizeof(void (*)())); // 分配存储函数指针的内存空间
    for (int i = 0; i < len; i++)
    {
        func_list[i] = func_table[index_list[i]]; // 根据索引获取对应的任务函数指针
    }
```

```
40.
41.     execute(func_list, len); // 执行任务序列
42.
43.     free(index_list); // 释放内存空间
44. }
45.
46. int main(int argc, char const *argv[])
47. {
48.     void (*func_table[])() = {task0, task1, task2, task3, task4, task5, task6, task7};
// 定义任务函数指针数组
49.     int *index_list;
50.     scheduler(index_list, func_table); // 调度任务序列
51.     return 0;
52. }
53.
```

运行程序：

```
~$ ./ex60307
56120653
task5 is called!
task6 is called!
task1 is called!
task2 is called!
task0 is called!
task6 is called!
task5 is called!
task3 is called!
~$
```

1.4 小结

本次实验我学到了：

1. 指针的声明、赋值、引用。
2. 用指针引用数组的元素。
3. 指向数组的指针。
4. 字符数组与字符串的使用。
5. 指针数组与字符指针数组的用法。
6. 指针函数与函数指针的用法。
7. 带有参数的 main 函数的用法。
8. ……

本次实验总的来说学到了很多，特别是 debug，虽然已经经历几个月的实验课，但还是几乎没有用不到 debug 的题，比较令我痛苦，感觉总是面向 bug 编程，而不是思路清晰一气呵成。虽然当今各种方便酷炫的 IDE、编辑器和 debug 工具大大方便了开发程序，但有时候，更好的能力基础还是得靠更笨的方法。我听说有的学校搞什么“人肉编辑器”“人肉编译器”，就是说用传统的原始的编辑器，甚至手写代码，来锻炼编写程序的能力。想必其中也是有一定的理念。但是平时我还是难以抗拒现代化编辑器方便的提示、补全、实时的语法警告以及自动排版等各种功能。

想想古早的程序员，他们的编辑器、编程语言甚至比手写 C 语言还要麻烦，我就不禁感到汗流浃背，他们需要在古老的终端中编写代码，没有智能提示、代码补全或者实时语法检查。每一行代码都需要亲手输入，每一个错误都需要花费大量的时间来排查和修复。但正是这些艰辛的经历锻炼出了那个时代的优秀程序员们，使他们在极端的环境下依然能够创造出令人惊叹的作品。

尽管我们现在拥有了许多先进的工具和技术，但是对于基本的编程能力和理解，仍然需要通过反复练习和不断的挑战来完善。现代化的编辑器和编程工具的确为我们提供了巨大的便利，但是我们也不能过分依赖它们。在适当的时候，还是要放下这些工具，用最基础的方式去编写和调试代码，这样才能更好地锻炼自己，提高自己的编程水平。

2 实验 7 结构与联合

2.1 表达式求值的程序验证

设有下面说明，请先自己计算表中表达式的值，然后通过编程计算来加以验证。各表达式之间相互无关。

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

```

2.1.1. 计算：

1. (++p)->x

分析：

1. ++p 返回(p+1)，即数组 a 第二个结构体元素首地址；
2. (++p)->x 即访问数组 a 中第二个结构体中的 x，其值为 100。

结果：100

2. p++, p->c

分析：

1. p++返回 p，并且 p 自增 1，即 p=p+1，此时 p 指向数组 a 第二个结构体元素首地址；
2. p->c 即访问数组 a 中第二个结构体中的 c，其值为'B'。

结果：'B'

3. *p++->t, *p->t

分析：

1. 分析*p++->t 中的运算符优先级，依次是++、->、*
2. p++使得 p 自增 1，即 p=p+1；
2. 分析*p->t 中的运算符优先级，依次是->、*；
4. p->t 即数组 a 中第 2 个结构体中的 t，其为一指向字符串 v 首地址的指针；
5. *p->t 即该字符串 v 的首个字符，即'x'。

结果：'x'

4. *(++p)->t

分析：

1. 分析*(++p)->t 中的运算符优先级，依次是()、++、->、*
2. ++p 返回(p+1)；
2. (++p)->t 即(p+1)->t，其为数组 a 中第 2 个结构体中的 t,为指向的字符串 v 首地址的指针；
4. *(++p)->t 即字符串 v 的首个字符，即'V'。

结果：'x'

5. *++p->t

分析：

1. 分析*++p->t 中的运算符优先级，依次是->、++、*；
2. p->t 即数组 a 中第 1 个结构体中的 t,为指向的字符串 u 首地址的指针；
3. ++p->t 返回字符串 u 第 2 个元素地址的指针；
4. *++p->t 即字符串 u 的第 2 个字符，即'x'。

结果：'V'

6. ++ * p->t

分析：

1. 分析++*p->t 中的运算符优先级，依次是->、*、++；
2. p->t 即数组 a 中第 1 个结构体中的 t,为指向的字符串 u 首地址的指针；
3. *p->t 返回字符串 u 第 1 个字符'U'；
4. ++*p->t 即字符串 u 的第 1 个元素自增 1，即'V'。

结果：'V'

2.1.2. 编程验证：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\1.exe
1
(++p)->x = 100
2
(p++, p->c) = B
3
(*p++->t, *p->t) = x
4
*(++p)->t = x
5
*++p->t = V
6
++ * p->t = V
```

2.1.3. 验证程序：

```c
#include <stdio.h>

int main()
{

    while (1)
    {
        char u[] = "UVWXYZ", v[] = "xyz";
        struct T
        {
            int x;
            char c;
            char *t;
        } a[] = {{11, 'A', u}, {100, 'B', v}}, *p = a;
        int id;
        scanf("%d", &id);
        switch (id)
        {
        case 1:
            printf("(++p)->x = %d\n", (++p)->x);
            break;
        case 2:
            printf("(p++, p->c) = %c\n", (p++, p->c));
            break;
        case 3:
            printf("(*p++->t, *p->t) = %c\n", (*p++->t, *p->t));
            break;
        case 4:
            printf("*(++p)->t = %c\n", *(++p)->t);
            break;
        case 5:
            printf("*++p->t = %c\n", *++p->t);
            break;
        case 6:
            printf("++ * p->t = %c\n", ++*p->t);
            break;
        default:
            break;
        }
    }

    return 0;
}

```

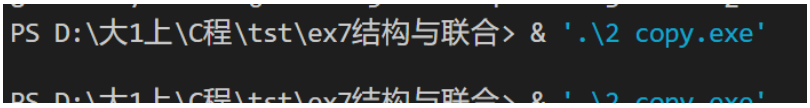
2.2 源程序修改替换

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

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

运行结果：

无输出



分析原因：

函数 create_list 对形参 headp 的修改不能作用于实参 head 上，导致头节点 head 始终是空结点。

程序修改：

```
1. /*实验 7-1 修改替换题:创建先进先出链表*/
2. #include <stdio.h>
3. #include <stdlib.h>
4. struct s_list
5. {
6.     int data;           /*数据域*/
7.     struct s_list *next; /*指针域*/
8. };
9. void create_list(struct s_list **headp, int *p);
10. int main(void)
11. {
12.     struct s_list *head = NULL, *p;
13.     int s[] = {1, 2, 3, 4, 5, 6, 7, 8, 0}; /*0 作为结束标记*/
14.     create_list(&head, s);
15.     p = head;
16.     while (p)
17.     {
18.         printf("%d\t", p->data);
19.         p = p->next;
20.     }
21.     printf("\n");
22.     return 0;
```

```
23. }
24. void create_list(struct s_list **headp, int *p)
25. {
26.     struct s_list *loc_head = NULL, *tail;
27.     if (p[0] == 0)
28.         ;
29.     else
30.     {
31.         loc_head = (struct s_list *)malloc(sizeof(struct s_list));
32.         loc_head->data = *p++;
33.         tail = loc_head;
34.         while (*p)
35.         {
36.             tail->next = (struct s_list *)malloc(sizeof(struct s_list));
37.             tail = tail->next;
38.             tail->data = *p++;
39.         }
40.         tail->next = NULL;
41.     }
42.     *headp = loc_head;
43. }
44.
```

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

```
1. /*实验 7-1 修改替换题:创建后进先出链表*/
2. #include <stdio.h>
3. #include <stdlib.h>
4. struct s_list
5. {
6.     int data;           /*数据域*/
7.     struct s_list *next; /*指针域*/
8. };
9. void create_list(struct s_list **headp, int *p);
10. int main(void)
11. {
12.     struct s_list *head = NULL, *p;
13.     int s[] = {1, 2, 3, 4, 5, 6, 7, 8, 0}; /*0 作为结束标记*/
14.     create_list(&head, s);
15.     p = head;
16.     while (p)
17.     {
18.         printf("%d\t", p->data);
```

```
19.         p = p->next;
20.     }
21.     printf("\n");
22.     return 0;
23. }
24. // 创建链表的函数定义
25. void create_list(struct s_list **headp, int *p)
26. {
27.     while (*p)
28.     {
29.         struct s_list *cur = (struct s_list *)malloc(sizeof(struct s_list)); // 分配内
存给新节点
30.         cur->data = *p++; // 设置新节点的数据为数组元素值
31.         
32.         if (*headp == NULL) // 如果链表为空，新节点就是头节点
33.         {
34.             cur->next = NULL; // 将新节点的 next 指针设为空
35.         }
36.         else // 如果链表不为空，将新节点连接到链表头部
37.         {
38.             cur->next = *headp; // 新节点的 next 指向原头节点
39.         }
40.         *headp = cur; // 更新头节点为新节点
41.     }
42. }
43.
```

运行结果：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\3.exe
8       7       6       5       4       3       2       1
PS D:\大1上\C程\tst\ex7结构与联合>
```

2.3 程序设计

2.3.1. 设计字段结构证

解题思路：

首先利用宏定义生成待调用函数，然后利用结构体联合体处理位操作，通过函数指针数组和输入整数的比特位状态来选择执行特定函数。核心在于动态确定函数调用，根据输入整数的位状态执行对应函数。

程序代码：

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

// 宏定义 TSK(id)，生成函数 f0 到 f7，用于打印函数被调用的信息
#define TSK(id) void f##id() { printf("the function %s is called!\n", #id); }

// 生成函数 f0 到 f7
TSK(0)
TSK(1)
TSK(2)
TSK(3)
TSK(4)
TSK(5)
TSK(6)
TSK(7)

// 定义包含八个位字段的结构体 Bits，用于表示一个字节的每个比特位
struct Bits
{
    unsigned char b0 : 1;
    unsigned char b1 : 1;
    unsigned char b2 : 1;
    unsigned char b3 : 1;
    unsigned char b4 : 1;
    unsigned char b5 : 1;
    unsigned char b6 : 1;
    unsigned char b7 : 1;
};

// 定义联合体 BITS，包含了一个无符号字符和一个 Bits 结构体，用于处理字节的位操作
typedef union
{
    unsigned char x;
    struct Bits octo;
} BITS;

int main(int argc, char const *argv[])
{
    // 定义函数指针数组 p[]，存储函数 f0 到 f7 的指针
    void (*p[])() = {f0, f1, f2, f3, f4, f5, f6, f7};

    // 用于存储输入的整数，表示字节的不同比特位状态
    int cmd_code;

```

```
45.     // 从标准输入读取一个整数，存储在 cmd_code 中
46.     scanf("%d", &cmd_code);
47.
48.     // 将输入的整数存储在 BITS 类型的变量 bits 中
49.     BITS bits = {cmd_code};
50.
51.     // 根据 bits 中每个比特位的值，选择调用 p[] 中相应索引位置的函数
52.     if (bits.octo.b0) p[0]();
53.     if (bits.octo.b1) p[1]();
54.     if (bits.octo.b2) p[2]();
55.     if (bits.octo.b3) p[3]();
56.     if (bits.octo.b4) p[4]();
57.     if (bits.octo.b5) p[5]();
58.     if (bits.octo.b6) p[6]();
59.     if (bits.octo.b7) p[7]();
60.
61.     // 程序执行完毕，返回 0
62.     return 0;
63. }
64.
```

运行程序：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\4.exe
123
the function 0 is called!
the function 1 is called!
the function 3 is called!
the function 4 is called!
the function 5 is called!
the function 6 is called!
PS D:\大1上\C程\tst\ex7结构与联合>
```

2.3.2. 班级成绩单

解题思路：

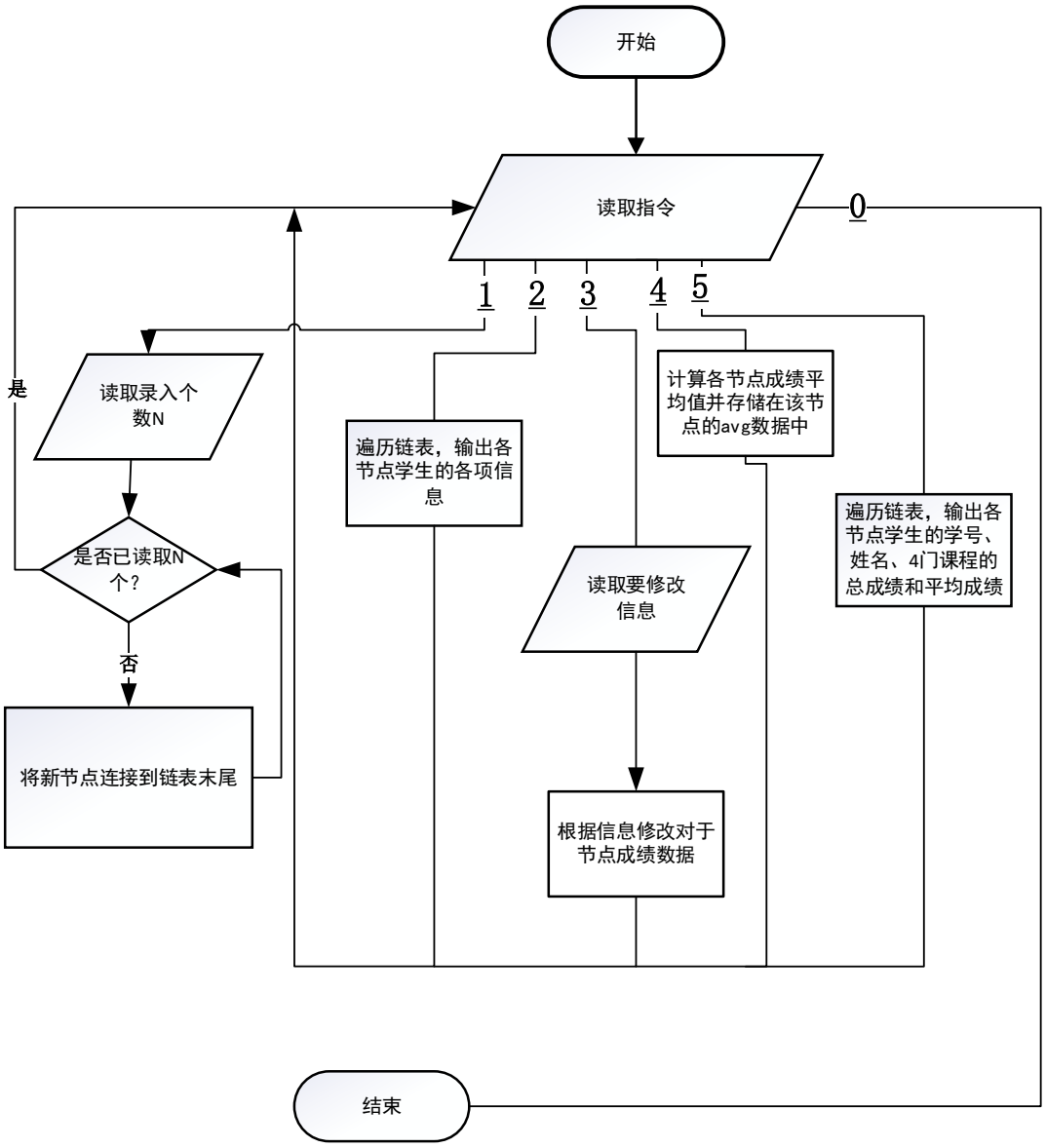


图 7-1 班级成绩管理系统流程图

程序代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct TEMP
{
    char uid[16];       // 学生 ID
    char name[16];      // 学生姓名
    int scores[4];      // 学生成绩数组，包含 4 门科目的成绩
    int sum;            // 学生成绩总和
```

```
11.     double avg;         // 学生成绩平均值
12.     struct TEMP *next;  // 下一个学生的指针
13. } STUDENTS;
14.
15. // 记录学生信息并返回指向学生结构的指针
16. STUDENTS *Record()
17. {
18.     STUDENTS *tmp = (STUDENTS *)malloc(sizeof(STUDENTS)); // 为学生结构分配内存
19.     tmp->next = NULL; // 设置下一个学生的指针为空
20.     // 从输入中读取学生信息
21.     scanf("%s %s %d %d %d %d", tmp->uid, tmp->name, &tmp->scores[0], &tmp->scores[1],
&tmp->scores[2], &tmp->scores[3]);
22.      tmp->sum = (tmp->scores[0] + tmp->scores[1] + tmp->scores[2] + tmp->scores[3]);
// 计算学生成绩总和
23.     tmp->avg = tmp->sum * .25; // 计算学生成绩平均值
24.     return tmp; // 返回指向学生结构的指针
25. }
26.
27. // 添加指定数量的新学生信息到链表中
28. void AddND(STUDENTS **students, int num)
29. {
30.     STUDENTS *cur = *students; // 当前学生
31.     STUDENTS *head = NULL, *tail; // 头结点和尾结点
32.     head = Record(); // 记录第一个学生信息并作为头结点
33.     tail = head; // 尾结点指向头结点
34.     // 循环添加新学生到链表
35.     for (int i = 1; i < num; i++)
36.     {
37.         STUDENTS *tmp = Record(); // 记录新学生信息
38.         tail->next = tmp; // 将新学生连接到链表尾部
39.         tail = tail->next; // 更新尾结点
40.     }
41.     // 将新的学生链表连接到已有链表或设置为新的链表
42.     if (*students != NULL)
43.     {
44.         while (cur->next != NULL)
45.         {
46.             cur = cur->next;
47.         }
48.         cur->next = head; // 将新学生链表连接到已有链表的尾部
49.     }
50.     else
51.     {
52.         *students = head; // 设置新的链表
```

```
53.     }
54. }
55.
56. // 计算学生链表中每个学生的平均成绩
57. void CalcAvg(STUDENTS *students)
58. {
59.     STUDENTS *cur = students; // 当前学生
60.     // 遍历链表计算每个学生的平均成绩
61.     while (cur != NULL)
62.     {
63.                 cur->sum  =  (cur->scores[0]  +  cur->scores[1]  +  cur->scores[2]  +
cur->scores[3]); // 计算学生成绩总和
64.         cur->avg = cur->sum * .25; // 计算学生成绩平均值
65.         cur = cur->next; // 移动到下一个学生
66.     }
67. }
68.
69. // 根据模式打印学生链表的信息
70. void PrintLN(STUDENTS *students, int mode)
71. {
72.     STUDENTS *cur = students; // 当前学生
73.     // 遍历链表并根据不同模式打印学生信息
74.     while (cur != NULL)
75.     {
76.         if (mode == 1)
77.         {
78.                 printf("%s %s %d %d %d %d\n", cur->uid, cur->name, cur->scores[0],
cur->scores[1], cur->scores[2], cur->scores[3]); // 打印学生详细信息
79.         }
80.         else if (mode == 2)
81.         {
82.             printf("%s %s %d %.2f\n", cur->uid, cur->name, cur->sum, cur->avg); // 打
印学生总成绩和平均成绩
83.         }
84.         cur = cur->next; // 移动到下一个学生
85.     }
86. }
87.
88. // 修改指定学生的指定科目成绩
89. void Modify(STUDENTS *students)
90. {
91.     char uid[16]; // 学生 ID
92.     int subject, score; // 科目和分数
93.     scanf("%s %d %d", uid, &subject, &score); // 读取输入的修改信息
```

```
94.     STUDENTS *cur = students; // 当前学生
95.     // 遍历链表查找要修改成绩的学生
96.     while (cur != NULL)
97.     {
98.         if (!strcmp(cur->uid, uid)) // 找到目标学生
99.         {
100.            cur->scores[subject - 1] = score; // 修改指定科目成绩
101.            return;
102.        }
103.        cur = cur->next; // 移动到下一个学生
104.    }
105. }
106.
107. int main(int argc, char const *argv[])
108. {
109.     int cmd; // 命令
110.     STUDENTS *students = NULL; // 学生链表
111.     while (1)
112.     {
113.         int num; // 数量
114.         scanf("%d", &cmd); // 读取命令
115.         switch (cmd) // 根据命令执行相应操作
116.         {
117.         case 1:
118.             scanf("%d", &num); // 读取要添加的学生数量
119.             AddND(&students, num); // 添加新学生到链表
120.             break;
121.         case 2:
122.             PrintLN(students, 1); // 打印学生详细信息
123.             break;
124.         case 3:
125.             Modify(students); // 修改学生信息
126.             CalcAvg(students); // 重新计算平均成绩
127.             break;
128.         case 4:
129.             CalcAvg(students); // 计算学生平均成绩
130.             break;
131.         case 5:
132.             PrintLN(students, 2); // 打印学生总成绩和平均成绩
133.             break;
134.         default:
135.             return 0; // 结束程序
136.             break;
137.         }
```

```
    }

    return 0;
}

```

运行程序：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\5.exe
1
1
U123456789 Elio 34 56 78 90
2
U123456789 Elio 34 56 78 90
3
U123456789 2 99
1
3
U987654321 Gray 0 0 0 0
U000000000 Black 100 100 100 100
U202073456 Red 45 34 67 99
5
U123456789 Elio 301 75.25
U987654321 Gray 0 0.00
U000000000 Black 400 100.00
U202073456 Red 245 61.25
0
PS D:\大1上\C程\tst\ex7结构与联合>
```

2.3.3. 回文字符串

解题思路：

本题关键在于回文链表判断：

- 使用快慢指针法找到链表中点，并在此过程中反转前半部分节点。
- 比较前半部分与后半部分节点的值，若均相等，则链表为回文结构。

程序代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct c_node
{
    char data;
    struct c_node *next;
} C_NODE;
```

```
10.
11. // 创建单链表
12. void createLinkList(C_NODE **headp, char s[])
13. {
14.     /********** 创建单链表 **********/
15.     C_NODE *cur = *headp = NULL; // 初始化当前节点和头节点为 NULL
16.     int i = 0;
17.     while (s[i] != '\0') // 遍历输入字符串
18.     {
19.         C_NODE *new_node = (C_NODE *)malloc(sizeof(C_NODE)); // 为新节点分配内存
20.         new_node->data = s[i]; // 设置新节点的数据为字符串中的字符
21.         new_node->next = NULL; // 将新节点的 next 指针设为 NULL
22.
23.         if (*headp == NULL) // 如果头节点为空，将新节点设置为头节点并更新当前节点
24.         {
25.             *headp = cur = new_node;
26.         }
27.         else // 否则将新节点连接到链表末尾
28.         {
29.             cur->next = new_node;
30.             cur = new_node;
31.         }
32.
33.         i++;
34.     }
35.     /********** 创建单链表 **********/
36. }
37.
38. // 判断是否为回文链表
39. void judgePalindrome(C_NODE *head)
40. {
41.     /********** 判断回文链表 **********/
42.     C_NODE *cur = head; // 初始化当前节点为头节点
43.     C_NODE *slow, *fast, *prev = NULL; // 初始化指针变量
44.     slow = fast = cur; // 初始化慢指针和快指针为头节点
45.     while (fast != NULL && fast->next != NULL) // 使用快慢指针找到链表中点
46.     {
47.         fast = fast->next->next;
48.         cur = slow->next;
49.         slow->next = prev;
50.         prev = slow;
51.         slow = cur;
52.     }
53.
```

```
    if (fast) // 如果链表节点数为奇数，调整慢指针位置
    {
        slow = slow->next;
    }
    while (slow != NULL) // 检查回文性质
    {
        if (slow->data != prev->data)
        {
            printf("false");
            return;
        }
        slow = slow->next;
        prev = prev->next;
    }
    printf("true");
    /********** 判断回文链表 **********/
}

int main()
{
    char s[1000], *pc = s; // 初始化输入字符串和指向字符串的指针
    int len = 0;
    C_NODE *head, *p; // 初始化链表头节点和遍历节点

    scanf("%[^\n]", s); // 读取输入字符串直到换行符为止

    createLinkList(&head, s); // 创建单链表

    for (p = head; p; p = p->next) // 计算单链表长度
        len++;
    if (len != strlen(s)) // 检查单链表长度与输入字符串长度是否一致
    {
        printf("单链表长度不正确");
        return 1;
    }
    else // 如果长度一致，检查单链表内容是否与输入字符串一致
    {
        for (p = head; p; p = p->next)
        {
            if (p->data != *pc++)
            {
                printf("单链表有错误结点");
                return 1;
            }
```

```
 98.         }
 99.     }
100.
101.     judgePalindrome(head); // 判断链表是否为回文链表
102.     return 0;
103. }
104.
```

运行程序：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\7_1.exe
race a car
false
PS D:\大1上\C程\tst\ex7结构与联合> .\7_1.exe
A man a plan aca nalp a nam A
true
PS D:\大1上\C程\tst\ex7结构与联合>
```

2.3.4. 成绩排序（一）

解题思路：

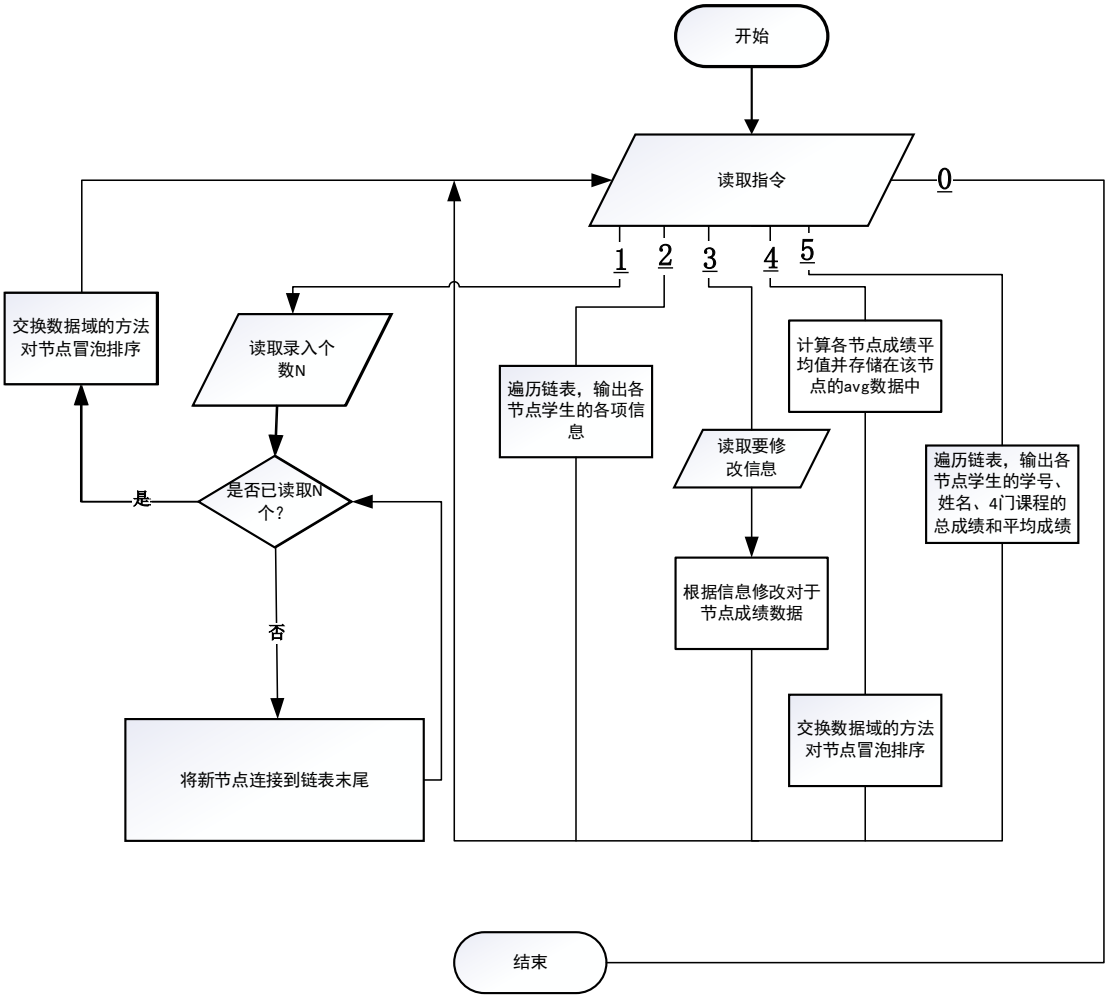


图 7-2 交换数据域方法成绩排序流程图

程序代码：

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

typedef struct TEMP
{
    char uid[16];      // 学生 ID
    char name[16];     // 学生姓名
    int scores[4];     // 学生成绩数组，包含 4 门科目的成绩
    int sum;           // 学生成绩总和
    double avg;        // 学生成绩平均值
    struct TEMP *next; // 下一个学生的指针
} STUDENTS;

// 记录学生信息并返回指向学生结构的指针
STUDENTS *Record()
{
    STUDENTS *tmp = (STUDENTS *)malloc(sizeof(STUDENTS)); // 为学生结构分配内存
    tmp->next = NULL;                                     // 设置下一个学生的指针为空
    // 从输入中读取学生信息
    scanf("%s %s %d %d %d %d", tmp->uid, tmp->name, &tmp->scores[0], &tmp->scores[1], &tmp->scores[2], &tmp->scores[3]);
     tmp->sum = (tmp->scores[0] + tmp->scores[1] + tmp->scores[2] + tmp->scores[3]); // 计算学生成绩总和
    tmp->avg = tmp->sum * .25;                                                       // 计算学生成绩平均值
    return tmp;                                                                      // 返回指向学生结构的指针
}

// 添加指定数量的新学生信息到链表中
void AddND(STUDENTS **students, int num)
{
    STUDENTS *cur = *students;    // 当前学生
    STUDENTS *head = NULL, *tail; // 头结点和尾结点
    head = Record();              // 记录第一个学生信息并作为头结点
    tail = head;                  // 尾结点指向头结点
    // 循环添加新学生到链表
    for (int i = 1; i < num; i++)
    {
        STUDENTS *tmp = Record(); // 记录新学生信息
```

```
38.         tail->next = tmp;         // 将新学生连接到链表尾部
39.         tail = tail->next;        // 更新尾结点
40.     }
41.     // 将新的学生链表连接到已有链表或设置为新的链表
42.     if (*students != NULL)
43.     {
44.         while (cur->next != NULL)
45.         {
46.             cur = cur->next;
47.         }
48.         cur->next = head; // 将新学生链表连接到已有链表的尾部
49.     }
50.     else
51.     {
52.         *students = head; // 设置新的链表
53.     }
54. }
55.
56. // 计算学生链表中每个学生的平均成绩
57. void CalcAvg(STUDENTS *students)
58. {
59.     STUDENTS *cur = students; // 当前学生
60.     // 遍历链表计算每个学生的平均成绩
61.     while (cur != NULL)
62.     {
63.                 cur->sum  =  (cur->scores[0]  +  cur->scores[1]  +  cur->scores[2]  +
cur->scores[3]); // 计算学生成绩总和
64.         cur->avg = cur->sum * .25;                                          //
计算学生成绩平均值
65.         cur = cur->next;                                                    //
移动到下一个学生
66.     }
67. }
68.
69. // 根据模式打印学生链表的信息
70. void PrintLN(STUDENTS *students, int mode)
71. {
72.     STUDENTS *cur = students; // 当前学生
73.     // 遍历链表并根据不同模式打印学生信息
74.     while (cur != NULL)
75.     {
76.         if (mode == 1)
77.         {
78.                 printf("%s %s %d %d %d %d\n", cur->uid, cur->name, cur->scores[0],
```

```
cur->scores[1], cur->scores[2], cur->scores[3]); // 打印学生详细信息
 79.         }
 80.         else if (mode == 2)
 81.         {
 82.             printf("%s %s %d %.2f\n", cur->uid, cur->name, cur->sum, cur->avg); // 打
印学生总成绩和平均成绩
 83.         }
 84.         cur = cur->next; // 移动到下一个学生
 85.     }
 86. }
 87.
 88. // 修改指定学生的指定科目成绩
 89. void Modify(STUDENTS *students)
 90. {
 91.     char uid[16];                           // 学生 ID
 92.     int subject, score;                     // 科目和分数
 93.     scanf("%s %d %d", uid, &subject, &score); // 读取输入的修改信息
 94.     STUDENTS *cur = students;               // 当前学生
 95.     // 遍历链表查找要修改成绩的学生
 96.     while (cur != NULL)
 97.     {
 98.         if (!strcmp(cur->uid, uid)) // 找到目标学生
 99.         {
100.             cur->scores[subject - 1] = score; // 修改指定科目成绩
101.             return;
102.         }
103.         cur = cur->next; // 移动到下一个学生
104.     }
105. }
106. // 交换两个学生节点的数据
107. void Swap(STUDENTS *cur, STUDENTS *post)
108. {
109.     // 交换平均成绩值
110.     double tmp = cur->avg;
111.     cur->avg = post->avg;
112.     post->avg = tmp;
113.
114.     // 交换总成绩值
115.     int tmp3 = cur->sum;
116.     cur->sum = post->sum;
117.     post->sum = tmp3;
118.
119.     // 交换姓名
120.     char tmp1[16];
```

```
121.     strcpy(tmp1, cur->name);
122.     strcpy(cur->name, post->name);
123.     strcpy(post->name, tmp1);
124.
125.     // 交换学生 ID
126.     strcpy(tmp1, cur->uid);
127.     strcpy(cur->uid, post->uid);
128.     strcpy(post->uid, tmp1);
129.
130.     // 交换学生科目成绩数组
131.     int tmp2[4];
132.     memcpy(tmp2, cur->scores, sizeof(int) * 4);
133.     memcpy(cur->scores, post->scores, sizeof(int) * 4);
134.     memcpy(post->scores, tmp2, sizeof(int) * 4);
135. }
136.
137. // 使用冒泡排序对学生链表按照平均成绩和姓名进行排序
138. void bubbleSort(STUDENTS *head)
139. {
140.     int swapped;
141.     STUDENTS *ptr;
142.     STUDENTS *end = NULL;
143.
144.     if (head == NULL)
145.     {
146.         return;
147.     }
148.
149.     do
150.     {
151.         swapped = 0;
152.         ptr = head;
153.
154.         while (ptr->next != end)
155.         {
156.             // 如果当前学生的平均成绩大于下一个学生的平均成绩，交换两个学生节点的数据
157.             if (ptr->avg > ptr->next->avg)
158.             {
159.                 Swap(ptr, ptr->next);
160.                 swapped = 1;
161.             }
162.             // 如果当前学生的平均成绩与下一个学生的平均成绩相等，按照姓名字典序排序
163.             else if (ptr->avg == ptr->next->avg)
164.             {
```

```
165.                 if (strcmp(ptr->name, ptr->next->name) > 0)
166.                 {
167.                     Swap(ptr, ptr->next);
168.                     swapped = 1;
169.                 }
170.             }
171.
172.             ptr = ptr->next;
173.         }
174.         end = ptr;
175.     } while (swapped);
176. }
177.
178. int main(int argc, char const *argv[])
179. {
180.     int cmd;                  // 命令
181.     STUDENTS *students = NULL; // 学生链表
182.
183.     while (1)
184.     {
185.         int num;          // 数量
186.         scanf("%d", &cmd); // 读取命令
187.
188.         switch (cmd) // 根据命令执行相应操作
189.         {
190.         case 1:
191.             scanf("%d", &num);     // 读取要添加的学生数量
192.             AddND(&students, num); // 添加新学生到链表
193.             bubbleSort(students);  // 对新加入的学生进行排序
194.             break;
195.         case 2:
196.             PrintLN(students, 1); // 打印学生详细信息
197.             break;
198.         case 3:
199.             Modify(students);  // 修改学生信息
200.             CalcAvg(students); // 重新计算平均成绩
201.             bubbleSort(students); // 对学生链表进行排序
202.             break;
203.         case 4:
204.             CalcAvg(students); // 计算学生平均成绩
205.             break;
206.         case 5:
207.             PrintLN(students, 2); // 打印学生总成绩和平均成绩
208.             break;
```

```
        default:
            return 0; // 结束程序
            break;
        }
    }

    return 0;
}

```

运行程序：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\6_2.exe
1
4
U202054321 Rose 89 94 85 100
U202056789 Tom 12 34 56 78
U202098765 Jerry 98 76 54 32
U202012345 Jack 98 76 54 32
2
U202056789 Tom 12 34 56 78
U202012345 Jack 98 76 54 32
U202098765 Jerry 98 76 54 32
U202054321 Rose 89 94 85 100
3
U202054321 1 66
4
5
U202056789 Tom 180 45.00
U202012345 Jack 260 65.00
U202098765 Jerry 260 65.00
U202054321 Rose 345 86.25
0
PS D:\大1上\C程\tst\ex7结构与联合> .\6_2.exe
```

2.3.5. 成绩排序（二）

解题思路：

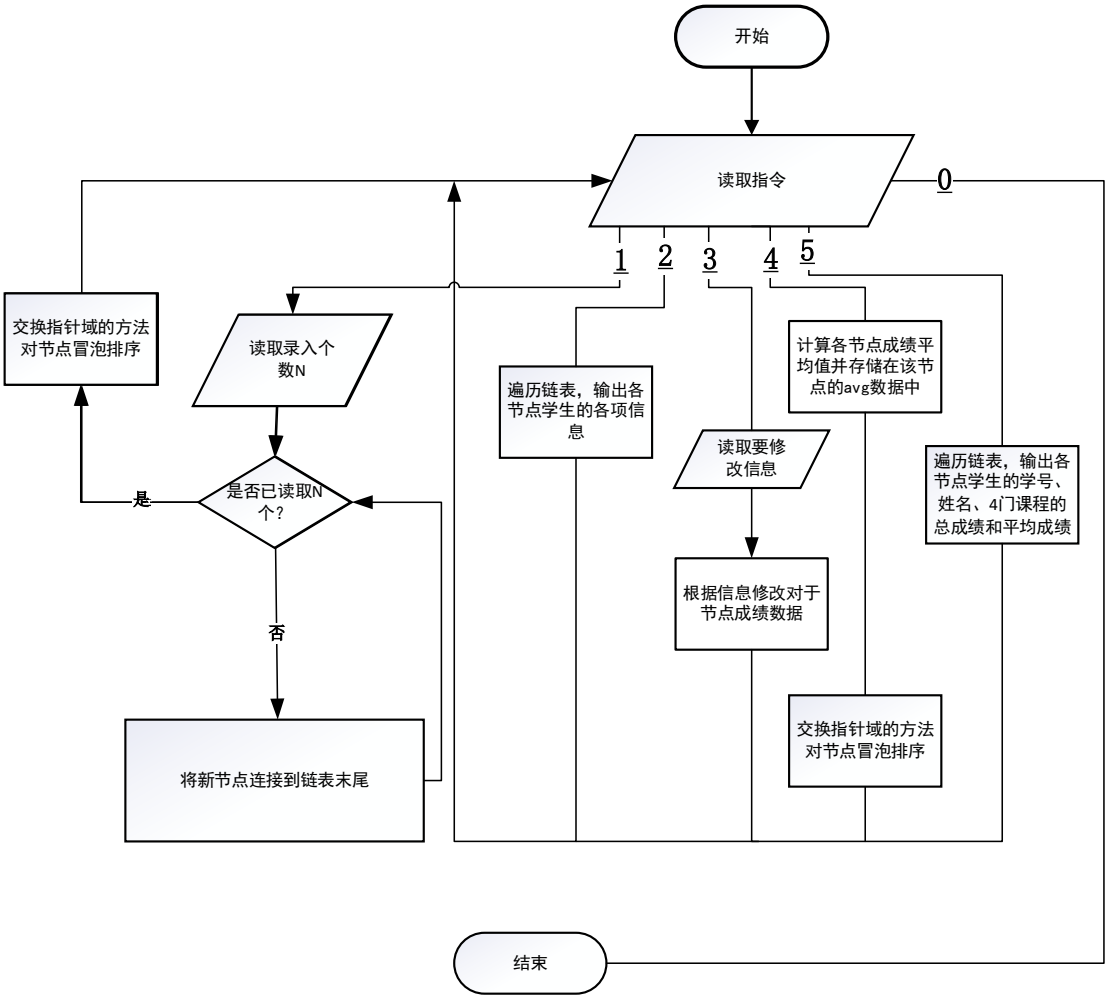


图 7-3 交换指针域方法成绩排序流程图

程序代码：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct TEMP
{
    char uid[16];      // 学生 ID
    char name[16];     // 学生姓名
    int scores[4];     // 学生成绩数组，包含 4 门科目的成绩
    int sum;           // 学生成绩总和
    double avg;        // 学生成绩平均值
    struct TEMP *next; // 下一个学生的指针
} STUDENTS;

// 记录学生信息并返回指向学生结构的指针
```

```
16. STUDENTS *Record()
17. {
18.     STUDENTS *tmp = (STUDENTS *)malloc(sizeof(STUDENTS)); // 为学生结构分配内存
19.     tmp->next = NULL;                                    // 设置下一个学生的指针为空
20.     // 从输入中读取学生信息
21.     scanf("%s %s %d %d %d %d", tmp->uid, tmp->name, &tmp->scores[0], &tmp->scores[1],
&tmp->scores[2], &tmp->scores[3]);
22.      tmp->sum = (tmp->scores[0] + tmp->scores[1] + tmp->scores[2] + tmp->scores[3]);
// 计算学生成绩总和
23.     tmp->avg = tmp->sum * .25;                                                  // 计
算学生成绩平均值
24.     return tmp;                                                                 // 返回
指向学生结构的指针
25. }
26.
27. // 添加指定数量的新学生信息到链表中
28. void AddND(STUDENTS **students, int num)
29. {
30.     STUDENTS *cur = *students;    // 当前学生
31.     STUDENTS *head = NULL, *tail; // 头结点和尾结点
32.     head = Record();              // 记录第一个学生信息并作为头结点
33.     tail = head;                  // 尾结点指向头结点
34.     // 循环添加新学生到链表
35.     for (int i = 1; i < num; i++)
36.     {
37.         STUDENTS *tmp = Record(); // 记录新学生信息
38.         tail->next = tmp;         // 将新学生连接到链表尾部
39.         tail = tail->next;        // 更新尾结点
40.     }
41.     // 将新的学生链表连接到已有链表或设置为新的链表
42.     if (*students != NULL)
43.     {
44.         while (cur->next != NULL)
45.         {
46.             cur = cur->next;
47.         }
48.         cur->next = head; // 将新学生链表连接到已有链表的尾部
49.     }
50.     else
51.     {
52.         *students = head; // 设置新的链表
53.     }
54. }
55.
```

```
56. // 计算学生链表中每个学生的平均成绩
57. void CalcAvg(STUDENTS *students)
58. {
59.     STUDENTS *cur = students; // 当前学生
60.     // 遍历链表计算每个学生的平均成绩
61.     while (cur != NULL)
62.     {
63.                 cur->sum  =  (cur->scores[0]  +  cur->scores[1]  +  cur->scores[2]  +
cur->scores[3]); // 计算学生成绩总和
64.         cur->avg = cur->sum * .25;                                                  //
计算学生成绩平均值
65.         cur = cur->next;                                                            //
移动到下一个学生
66.     }
67. }
68.
69. // 根据模式打印学生链表的信息
70. void PrintLN(STUDENTS *students, int mode)
71. {
72.     STUDENTS *cur = students; // 当前学生
73.     // 遍历链表并根据不同模式打印学生信息
74.     while (cur != NULL)
75.     {
76.         if (mode == 1)
77.         {
78.                 printf("%s %s %d %d %d %d\n", cur->uid, cur->name, cur->scores[0],
cur->scores[1], cur->scores[2], cur->scores[3]); // 打印学生详细信息
79.         }
80.         else if (mode == 2)
81.         {
82.             printf("%s %s %d %.2f\n", cur->uid, cur->name, cur->sum, cur->avg); // 打
印学生总成绩和平均成绩
83.         }
84.         cur = cur->next; // 移动到下一个学生
85.     }
86. }
87.
88. // 修改指定学生的指定科目成绩
89. void Modify(STUDENTS *students)
90. {
91.     char uid[16];                           // 学生 ID
92.     int subject, score;                     // 科目和分数
93.     scanf("%s %d %d", uid, &subject, &score); // 读取输入的修改信息
94.     STUDENTS *cur = students;               // 当前学生
```

```
95.     // 遍历链表查找要修改成绩的学生
96.     while (cur != NULL)
97.     {
98.         if (!strcmp(cur->uid, uid)) // 找到目标学生
99.         {
100.            cur->scores[subject - 1] = score; // 修改指定科目成绩
101.            return;
102.        }
103.        cur = cur->next; // 移动到下一个学生
104.    }
105. }
106. // 使用节点指针域调整链表顺序进行升序排序
107. void bubbleSort(STUDENTS **head) {
108.     int swapped;
109.     STUDENTS *ptr;
110.     STUDENTS *end = NULL;
111.
112.     if (*head == NULL) {
113.         return;
114.     }
115.
116.     do {
117.         swapped = 0;
118.         ptr = *head;
119.
120.         while (ptr->next != end) {
121.             // 如果当前学生的平均成绩大于下一个学生的平均成绩，交换节点的指针域连接方式
122.                     if (ptr->avg > ptr->next->avg ||(ptr->avg == ptr->next->avg &&
strcmp(ptr->name, ptr->next->name) > 0) ) {
123.                 STUDENTS *tmp = ptr->next;
124.                 ptr->next = tmp->next;
125.                 tmp->next = ptr;
126.
127.                 if (ptr == *head) {
128.                     *head = tmp;
129.                 } else {
130.                     // 找到前一个节点，并连接到交换后的节点
131.                     STUDENTS *prev = *head;
132.                     while (prev->next != ptr) {
133.                         prev = prev->next;
134.                     }
135.                     prev->next = tmp;
136.                 }
137.
```

```
138.                 swapped = 1;
139.             }
140.             if (!swapped)
141.             {
142.                 ptr = ptr->next;
143.             }
144.         }
145.         end = ptr;
146.     } while (swapped);
147. }
148.
149. int main(int argc, char const *argv[])
150. {
151.     int cmd;                  // 命令
152.     STUDENTS *students = NULL; // 学生链表
153.
154.     while (1)
155.     {
156.         int num;          // 数量
157.         scanf("%d", &cmd); // 读取命令
158.
159.         switch (cmd) // 根据命令执行相应操作
160.         {
161.         case 1:
162.             scanf("%d", &num);     // 读取要添加的学生数量
163.             AddND(&students, num); // 添加新学生到链表
164.             bubbleSort(&students);  // 对新加入的学生进行排序
165.             break;
166.         case 2:
167.             PrintLN(students, 1); // 打印学生详细信息
168.             break;
169.         case 3:
170.             Modify(students);  // 修改学生信息
171.             CalcAvg(students); // 重新计算平均成绩
172.             bubbleSort(&students); // 对学生链表进行排序
173.             break;
174.         case 4:
175.             CalcAvg(students); // 计算学生平均成绩
176.             break;
177.         case 5:
178.             PrintLN(students, 2); // 打印学生总成绩和平均成绩
179.             break;
180.         default:
181.             return 0; // 结束程序
```

```
182.             break;
183.         }
184.     }
185.
186.     return 0;
187. }
188.
```

运行程序：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\8_1.exe
1
4
U202054321 Rose 89 94 85 100
U202056789 Tom 12 34 56 78
U202098765 Jerry 98 76 54 32
U202012345 Jack 98 76 54 32
5
U202056789 Tom 180 45.00
U202012345 Jack 260 65.00
U202098765 Jerry 260 65.00
U202054321 Rose 368 92.00
2
U202056789 Tom 12 34 56 78
U202012345 Jack 98 76 54 32
U202098765 Jerry 98 76 54 32
U202054321 Rose 89 94 85 100
3
U202054321 1 66
4
5
U202056789 Tom 180 45.00
U202012345 Jack 260 65.00
U202098765 Jerry 260 65.00
U202054321 Rose 345 86.25
0
PS D:\大1上\C程\tst\ex7结构与联合>
```

2.3.6. 逆波兰表达式

解题思路：

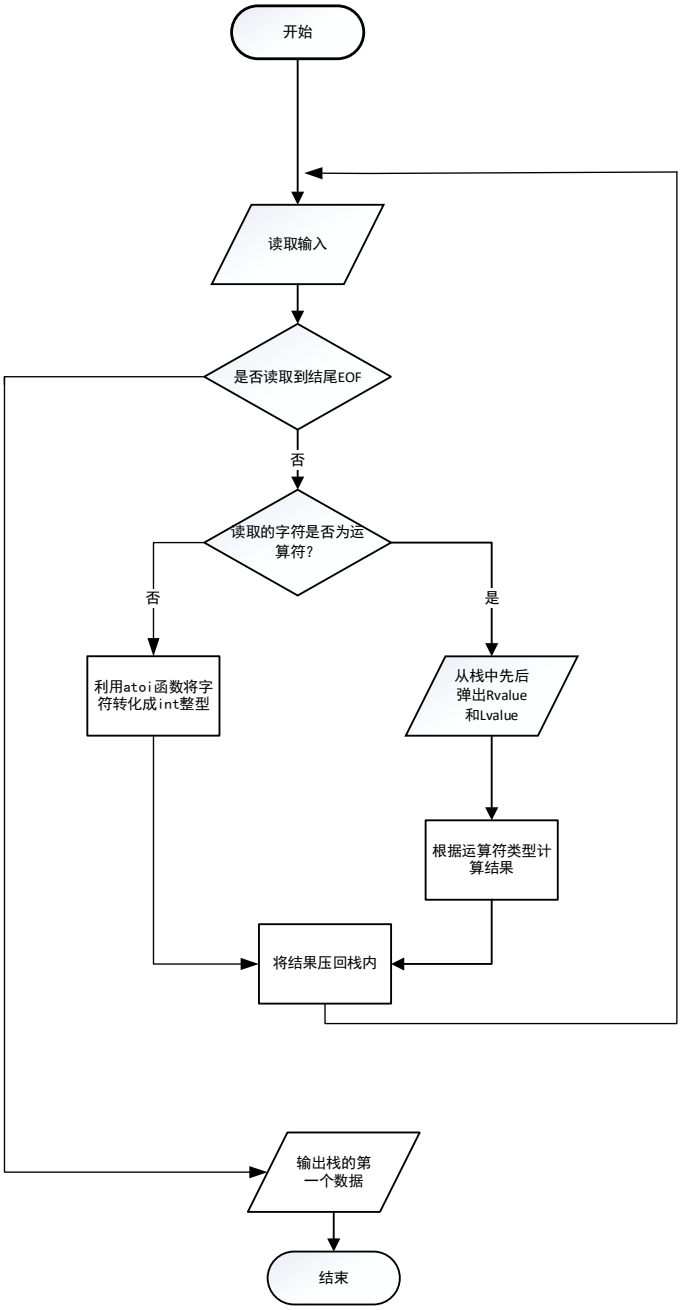


图 7-4 逆波兰表达式流程图

程序代码：

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <ctype.h>
5.
6. // 定义一个结构体，用于表示数字和操作符
7. typedef struct TEMP
```

```
 8. {
 9.     int num;           // 数字或结果
10.     struct TEMP *next;  // 指向下一个结点的指针
11. } TOKEN;
12.
13. // 将元素压入栈中
14. void Push(TOKEN **stack, int value)
15. {
16.     TOKEN *cur = (TOKEN *)malloc(sizeof(TOKEN));  // 分配内存以存储新元素
17.     cur->num = value;  // 将值存储到新元素中
18.
19.     // 如果栈为空，则新元素就是栈顶元素
20.     if (*stack == NULL)
21.     {
22.         cur->next = NULL;  // 栈顶元素的下一个为空
23.         *stack = cur;      // 更新栈顶指针为新元素
24.         return;
25.     }
26.     else
27.     {
28.         cur->next = *stack;  // 新元素的下一个是当前栈顶元素
29.         *stack = cur;        // 更新栈顶指针为新元素
30.     }
31. }
32.
33. // 弹出栈顶元素
34. int Pop(TOKEN **stack)
35. {
36.     int value = (*stack)->num;  // 获取栈顶元素的值
37.     *stack = (*stack)->next;    // 更新栈顶指针为下一个元素
38.     return value;               // 返回弹出的值
39. }
40.
41. // 检查是否为数字
42. int IsNum(char *seg)
43. {
44.     if (strlen(seg) == 1)  // 如果长度为 1
45.     {
46.         if (isdigit(*seg))  // 如果是数字字符
47.         {
48.             return 1;  // 返回 1 表示是数字
49.         }
50.         else
51.         {
```

```
52.             return 0;  // 返回 0 表示不是数字
53.         }
54.     }
55.     else
56.     {
57.         return 1;  // 如果长度不为 1，假定是数字
58.     }
59. }
60.
61. int main(int argc, char const *argv[])
62. {
63.     TOKEN *stack = NULL;  // 初始化栈为空
64.     char seg[8], operater;  // 存储输入的字符串和操作符
65.     int value, product, lvalue, rvalue;  // 存储值和计算结果
66.
67.     // 逐个读取输入的字符串
68.     while (scanf("%s", seg) != EOF)
69.     {
70.         if (!IsNum(seg))  // 如果不是数字，即为操作符
71.         {
72.             rvalue = Pop(&stack);  // 弹出栈顶元素作为右操作数
73.             lvalue = Pop(&stack);  // 弹出栈顶元素作为左操作数
74.
75.             switch (*seg)  // 根据操作符进行计算
76.             {
77.             case '+':
78.                 product = lvalue + rvalue;
79.                 break;
80.             case '-':
81.                 product = lvalue - rvalue;
82.                 break;
83.             case '*':
84.                 product = lvalue * rvalue;
85.                 break;
86.             case '/':
87.                 product = lvalue / rvalue;
88.                 break;
89.             default:
90.                 break;
91.             }
92.
93.             Push(&stack, product);  // 将计算结果压入栈中
94.         }
95.         else  // 如果是数字
```

```
96.         {
97.             value = atoi(seg);  // 将字符串转换为整数
98.             Push(&stack, value);  // 将数字压入栈中
99.         }
100.    }
101.
102.    printf("%d", stack->num);  // 打印最终结果
103.    return 0;
104. }
105.
```

运行程序：

```
PS D:\大1上\C程\tst\ex7结构与联合> .\9 1.exe
4  13  5  /  +
^Z
6
PS D:\大1上\C程\tst\ex7结构与联合>
```

2.4 小结

在本次实验中，我学到了：

1. 结构的说明和引用、结构的指针、结构数组以及函数中使用结构的方法
2. 动态分配函数 malloc、calloc 的用法
3. 自引用结构以及单向链表的创建、遍历、结点的增删、查找等操作。
4. 字段结构和联合的用法。

由于 C 语言是我接触的第一门较为底层的语言，之前对结构、联合从未有过了解，初上机便感到了链表的抽象：一开始连创建列表的函数都写不出来！但经过理论学习，以及踩了各种坑后，各种概念在脑中愈发清晰，逐渐对链表操作更加得心应手，切实体会到了知识的巧妙和自身的进步。

在写这篇实验报告时，又把有关的题目敲了一遍，发现并改进了之前写的程序的诸多不必要操作。比如创建链表的函数，先进先出与先进后出所需的中间变量的数量差异，如何更快的交换两个结点的数据域、指针域等等，使我对这一章的内容的理解更加深入。

在写程序时，印象最深的莫过指针域的交换，前后要用到众多中间变量，相比数组元素交换麻烦许多。同时，在交换数据域时，也遇到了如何快速交换数组的问题，由于定义数组时，指向数组的标识符类型为只读，因此不能直接交换，需要另设两个二级指针分别指向两

个数组的指针，通过这个二级指针来交换两个数组标识符指向的首地址，从而达成目的。

经过这一学期的 C 语言学习，自认还算学的比较认真，收获也很多，但仅课本上就仍有数不清的细节我还没掌握。切实体会到了 C 语言厚重的历史沉淀。

感谢学院，感谢老师的教导，感谢助教一学期来对我的帮助，让我能在这半年中始终保持对 C 语言的热情。C 语言值得每个计算机学生认真研究。

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

[2] 华中科技大大学计算机学院.《C 语言程序设计实验》报告要求及格式参考.华中科技大大学,2023

[3] 卢萍,李开,王多强，甘早斌. C 语言程序设计,北京：清华大学出版社,2019

[4] Arman Hilmioglu. Easy Code Formatter for Word,Microsoft AppSource, 2023.https://appsource.microsoft.com/en-us/product/office/wa104382008

[5] Arman Hilmioglu. Easy Code Formatter Style. https://github.com/armhil/easy-code-formatter-styles