

# 数据结构与算法分析

## 实验报告



姓 名
班 级
实 验 名 称
开 设 学 期
开 设 时 间
报 告 日 期
评 定 成 绩

东北大学软件学院

---

# 实验报告

课程名称：数据结构与算法分析 班级：软信 2001 实验成绩：  
实验名称：顺序表和链表的应用 学号：20207130 批阅教师签字：  
实验编号：实验一 姓名：赖骏鸿 实验日期：2022 年 5 月 12 日  
指导教师：马毅 组号：暂无 实验时间：18 时 30 分—22 时 10 分

## 一、实验目的

- 掌握线性表的基本操作（插入、删除、查找）以及线性表合并等运算在顺序存储结构、链式存储结构上的实现。重点掌握链式存储结构实现的各种操作。
- 掌握线性表的链式存储结构的应用。

## 二、实验内容与实验步骤

### 1. 实验内容：

长整数四则运算（难度系数为 5，记作实验 1.1），Joseph 环（难度系数为 3，记作实验 1.2）

#### 实验 1.1：

##### [问题描述]

设计一个实现任意长的整数进行加法运算的演示程序。

##### [基本要求]

利用双向循环链表实现长整数的存储，每个节点包含一个整型变量。任何整型变量的范围是 $-(2^{15}-1) \sim 2^{15}-1$ ，输入输出形式：按照中国对于长整数的表示习惯，每四位一组，组间用逗号分开。

##### [测试数据]

- (1) 0; 0; 应输入“0”
- (2) -2345,6789;-7654,3211;应输出“-1,0000,0000”
- (3) 1,0001,0001;-1,0001,0001;应输出 0。

##### [实现提示]

每个结点中可以存放的最大整数为  $2^{15}-1=32768$ ，才能保证两数相加不会溢出。但若这样存放，即相当于按 32768 进制数存放，在十进制数与 32768 进制数之间的转换十分不便。故可以在每个节点中仅存放十进制数的 4 位，即不超过 9999 的非负整数，整个链表表示为万进制数。

可以利用头节点的数据域的符号代表长整数的符号。相加过程中不要破坏两个操作数链表。不能给长整数位数规定上限。

### 实验 1.2：

##### [问题描述]

约瑟夫环（Joseph）问题的一种描述是：编号为 1、2、3……n 的 n 个人按照顺时针方向围坐一圈，每人持有一个密码（正整数）。一开始任选一个正整数作为报数的上限值 m，从第一个人开始按照顺时针的方向自 1 开始顺序报数，报到 m 时停止报数。报 m 的人出列，将他的密码作为新的 m 值，从他的顺时针方向上的下一个人开始重新从 1 报数，如此下去，直至所有人全部出列为止。试设计一个程序求出出列顺序。

##### [基本要求]

利用顺序表、单向循环链表两种存储结构模拟此过程，按照出列的顺序依次输出每个人的编号。

#### [测试数据]

m 的初值为 20, n=7, 7 个人的密码依次为：3、1、7、2、4、8、4，首先 m 值为 6（正确的出列顺序应为 6、1、4、7、2、3、5）。

#### [实现提示]

程序运行后，首先要求用户指定初始报数上限值，然后读取个人的密码。可设  $n \leq 30$ 。  
此题所用的循环链表中不需要“头结点”，请注意空表和非空表的界限。

### 2. 数据结构及函数说明

## 实验 1.1：

#### (1) 自定义结构体

1) 可以存储任意类型的自定义基本数据类型

```
typedef struct Elemtpe{  
    void* type;  
}Elemtpe;
```

2) 双向循环链表

```
typedef struct DCLNode{  
    Elemtpe data;  
    struct DCLNode* next;  
    struct DCLNode* pre;  
}DCLNode,*DCLinkedlist;
```

#### (2) 函数说明

```
int Init_list(DCLinkedlist &list,char* input);  
void Display(const DCLinkedlist &list);  
int length(DCLinkedlist &list);  
int size_cmp(DCLinkedlist &a,DCLinkedlist &b);  
void sub_list(DCLinkedlist &a,DCLinkedlist &b);  
void add_list(DCLinkedlist &a,DCLinkedlist &b);  
void run_caculate(DCLinkedlist &a,DCLinkedlist &b);
```

int Init\_list(DCLinkedlist &list,char\* input);：根据字符串初始化对应的双向循环链表  
void Display(const DCLinkedlist &list);打印对应的双向循环链表信息  
int length(DCLinkedlist &list);计算双向循环链表的长度  
int size\_cmp(DCLinkedlist &a,DCLinkedlist &b);比较双向循环链表 a, b 所存储的数值大小  
void sub\_list(DCLinkedlist &a,DCLinkedlist &b);对双向循环链表 a, b 进行减法操作  
void add\_list(DCLinkedlist &a,DCLinkedlist &b);双向循环链表 a, b 进行加法操作  
void run\_caculate(DCLinkedlist &a,DCLinkedlist &b);根据输入数据的符号判断分别进行加法操作

### 3. 实验思路

首先在函数中初始化好对应的两个待测试的字符串 caa 和 cba，字符串必须是题目设定

的标准形式。然后声明双向循环链表 a, b, 调用函数 Init\_list 对链表和字符串进行初始化。初始化完成后，调用函数 run\_caculate 进行加法操作，其中 run\_caculate 函数内部结构包含了 add\_list 方法和 sub\_list 方法。最后调用函数 Display 打印出实验结果。

#### 4. 实验流程图



#### 5. 实验源代码

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

typedef struct Elemtyp{
    void* type;
}Elemtyp;
typedef struct DCLNode{
    Elemtyp data;
    struct DCLNode* next;
    struct DCLNode* pre;
}DCLNode,*DCLinkedlist;
int Init_list(DCLinkedlist &list,char* input);
void Display(const DCLinkedlist &list);
int length(DCLinkedlist &list);
int size_cmp(DCLinkedlist &a,DCLinkedlist &b);
void sub_list(DCLinkedlist &a,DCLinkedlist &b);
void add_list(DCLinkedlist &a,DCLinkedlist &b);
```

```
void run_caculate(DCLinkedlist &a,DCLinkedlist &b);

int Init_list(DCLinkedlist &list,char* input){
    list=(DCLinkedlist)malloc(sizeof(DCLNode));
    list->next= NULL;
    list->pre=NULL;
    if(list==NULL){
        return 0;
    }
    int i=0;
    DCLNode* p=list;
    if(input[i]=='-'){
        Elemtpe e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=1;
        list->data=e;
        i++;
    }else{
        Elemtpe e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=0;
        list->data=e;
    }
    while(input[i]!='\0'){
        int index=0;
        char temp[5]="";
        //char temp[5];
        while(input[i]!=','&&input[i]!='\0'){
            temp[index++]=input[i++];
        }
        int data=atoi(temp);
        DCLinkedlist newnode=(DCLinkedlist)malloc(sizeof(DCLNode));
        newnode->next=NULL;
        newnode->pre=NULL;
        Elemtpe e;
        e.type=(int*)malloc(sizeof(int));
        // if(*(int*)list->data.type==1){
        //     data=-data;
        // }
        *(int*)e.type=data;
        newnode->data=e;
        p->next=newnode;
        newnode->pre=p;
        p=newnode;
    }
}
```

```

    if(input[i]==' '){
        i++;
        continue;
    }
    if(input[i]=='\0'){
        p->next=list;
        list->pre=p;
        break;
    }
}

return 1;
}

void Display(const DCLinkedlist &list){
    DCLinkedlist p,q;
    p=list;
    q=list;
    p=p->next;
    q=q->next;
    int sum=0;
    while(p!=list){
        sum=sum+*(int*)p->data.type;
        p=p->next;
    }
    if(sum==0){
        printf("%d",sum);
        return;
    }
    if(*(int*)list->data.type==1){
        printf("- ");
    }
    while(q->next!=list){
        if(*(int*)q->data.type==0){
            printf("0000,");
        }else{
            if(list->next!=q){
                int times=0;
                if(0<*(int*)q->data.type&&*(int*)q->data.type<10){
                    times=3;
                }
                if(9<*(int*)q->data.type&&*(int*)q->data.type<100){
                    times=2;
                }
                if(99<*(int*)q->data.type&&*(int*)q->data.type<1000){

```

```

        times=1;
    }
    for(int index=0;index<times;index++){
        printf("0");
    }
}
printf("%d",*(int*)q->data.type);
printf(",");
}
q=q->next;
}

if(*(int*)q->data.type==0){
    printf("0000");
}else{
    int times=0;
    if(0<*(int*)q->data.type&&*(int*)q->data.type<10){
        times=3;
    }
    if(9<*(int*)q->data.type&&*(int*)q->data.type<100){
        times=2;
    }
    if(99<*(int*)q->data.type&&*(int*)q->data.type<1000){
        times=1;
    }
    for(int index=0;index<times;index++){
        printf("0");
    }
    printf("%d",*(int*)q->data.type);
}
}

int length(DCLinkedlist &list){
    DCLinkedlist p=list;
    p=p->next;
    int leng=0;
    while(p!=list){
        leng++;
        p=p->next;
    }
    return leng;
}

int size_cmp(DCLinkedlist &a,DCLinkedlist &b){
    if(length(a)!=length(b)){
        return 0;
    }else{

```

```

DCLinkedlist starta=a->next;
DCLinkedlist startb=b->next;
while(starta!=a&&startb!=b){
    if(*(int*)starta->data.type<*(int*)startb->data.type){
        return 0;
    }else{
        return 1;
    }
}
return 1;
}

void sub_list(DCLinkedlist &a,DCLinkedlist &b){
    DCLinkedlist enda=a->pre;
    DCLinkedlist endb=b->pre;
    int borrow=0;
    if(length(a)>length(b)||size_cmp(a,b)){
        while(enda!=a&&endb!=b){
            *(int*)enda->data.type=*(int*)enda->data.type-*(int*)endb->data.type-borrow;
            borrow=0;
            if(*(int*)enda->data.type<0){
                *(int*)enda->data.type=*(int*)enda->data.type+10000;
                borrow=1;
            }
            enda=enda->pre;
            endb=endb->pre;
        }
        while(enda!=a){
            *(int*)enda->data.type=*(int*)enda->data.type-borrow;
            borrow=0;
            if(*(int*)enda->data.type<0){
                *(int*)enda->data.type=*(int*)enda->data.type+10000;
                borrow=1;
            }
            enda=enda->pre;
        }

        while(*(int*)a->next->data.type==0&&a->next->next!=a){
            a->next=a->next->next;
            a->next->pre=a;
        }
        Display(a);
    }
    if(length(a)<length(b)||size_cmp(b,a)){

```

```

while(enda!=a&&endb!=b){
    *(int*)endb->data.type=*(int*)endb->data.type-*(int*)enda->data.type-borrow;
    borrow=0;
    if(*(int*)endb->data.type<0{
        *(int*)endb->data.type=*(int*)endb->data.type+10000;
        borrow=1;
    }
    enda=enda->pre;
    endb=endb->pre;
}
while(endb!=b){
    *(int*)endb->data.type=*(int*)endb->data.type-borrow;
    borrow=0;
    if(*(int*)endb->data.type<0{
        *(int*)endb->data.type=*(int*)endb->data.type+10000;
        borrow=1;
    }
    endb=endb->pre;
}

while(*(int*)b->next->data.type==0&&b->next->next!=b){
    b->next=b->next->next;
    b->next->pre=b;
}
Display(b);
}

void add_list(DCLinkedlist &a,DCLinkedlist &b){
    DCLinkedlist enda=a->pre;
    DCLinkedlist endb=b->pre;
    int carry=0;
    if(length(a)>=length(b)){
        while(enda!=a&&endb!=b){
            *(int*)enda->data.type=*(int*)enda->data.type+*(int*)endb->data.type+carry;
            carry=0;
            if(*(int*)enda->data.type>=10000){
                *(int*)enda->data.type=*(int*)enda->data.type-10000;
                carry=1;
            }
            enda=enda->pre;
            endb=endb->pre;
        }
        while(enda!=a){
            *(int*)enda->data.type=*(int*)enda->data.type+carry;
        }
    }
}

```

```

carry=0;
if(*(int*)enda->data.type>=10000){
    *(int*)enda->data.type=*(int*)enda->data.type-10000;
    carry=1;
}
enda=enda->pre;
}
if(carry==1){
    DCLinkedlist newnode=(DCLinkedlist)malloc(sizeof(DCLNode));
    newnode->next=NULL;
    newnode->pre=NULL;
    Elemtyp e;
    e.type=(int*)malloc(sizeof(int));
    *(int*)e.type=carry;
    newnode->data=e;
    newnode->next=a->next;
    a->next->pre=newnode;
    newnode->pre=a;
    a->next=newnode;
}
Display(a);
}else{
while(enda!=a&&endb!=b){
    *(int*)endb->data.type=*(int*)enda->data.type+*(int*)endb->data.type+carry;
    carry=0;
    if(*(int*)endb->data.type>=10000){
        *(int*)endb->data.type=*(int*)endb->data.type-10000;
        carry=1;
    }
    enda=enda->pre;
    endb=endb->pre;
}
while(endb!=b){
    *(int*)endb->data.type=*(int*)endb->data.type+carry;
    carry=0;
    if(*(int*)endb->data.type>=10000){
        *(int*)endb->data.type=*(int*)endb->data.type-10000;
        carry=1;
    }
    endb=endb->pre;
}
if(carry==1){
    DCLinkedlist newnode=(DCLinkedlist)malloc(sizeof(DCLNode));
    newnode->next=NULL;
}
}

```

```

    newnode->pre=NULL;
    Elemtyp e;
    e.type=(int*)malloc(sizeof(int));
    *(int*)e.type=carry;
    newnode->data=e;
    newnode->next=b->next;
    b->next->pre=newnode;
    newnode->pre=b;
    b->next=newnode;
}
Display(b);
}
}

void run_caculate(DCLinkedlist &a,DCLinkedlist &b){
    if(*(int*)a->data.type==*(int*)b->data.type){
        add_list(a,b);
    }else{
        sub_list(a,b);
    }
}

int main(){
    DCLinkedlist a;
    DCLinkedlist b;
    char caa[]={-34,5367};
    char cba[]={-5,5433};
    char* ca=&caa[0];
    char* cb=&cba[0];
    Init_list(a,ca);
    Init_list(b,cb);
    run_caculate(a,b);
    system("pause");
}

```

## 实验 1.2:

### 1. 自定义结构体

#### (1) 自定义链表

```

typedef struct CLnode{
    int id;
    int key;
    struct CLnode* next;
}CLnode,*CLinklist;

```

## 2. 函数说明

```
void Create_Empty_CL(CLinklist &CL);
void Insert_Head_CL(int elem, CLinklist &CL, int pos);
void Insert_Tail_CL(int elem, CLinklist &CL, int idn);
void Display_CL(CLinklist &CL);
CLnode* Solution_Joseph(int m, int numperson, int* keyperson, CLinklist &CL);
```

void Create\_Empty\_CL(CLinklist &CL);创建空链表

void Insert\_Head\_CL(int elem, CLinklist &CL, int pos);根据入口参数插入新节点到链表

void Insert\_Tail\_CL(int elem, CLinklist &CL, int idn);链表的尾部插入实现

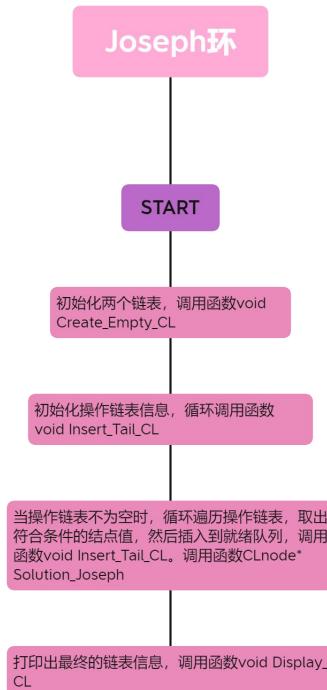
void Display\_CL(CLinklist &CL);打印链表信息

CLnode\* Solution\_Joseph(int m, int numperson, int\* keyperson, CLinklist &CL);主逻辑函数

## 3. 实验思路

首先初始化两个链表，操作队列和就绪队列。根据系统数据初始化操作队列，就绪队列为空。初始化计数器等于 0，当操作队列不为空时循环操作，将符合条件的结点新插入到就绪队列中。调用函数 CLnode\* Solution\_Joseph 返回就绪队列，在主函数中调用函数 void Display\_CL 即可打印出就绪队列信息，即为结果。

## 4. 实验流程图



## 5. 源代码

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

typedef struct CLinklist{
    int id;
    int key;
```

```

    struct CLnode* next;
}CLnode,*CLinklist;
void Create_Empty_CL(CLinklist &CL);
void Insert_Head_CL(int elem,CLinklist &CL,int pos);
void Insert_Tail_CL(int elem,CLinklist &CL,int idn);
void Display_CL(CLinklist &CL);
CLnode* Solution_Joseph(int m,int numperson,int* keyperson,CLinklist &CL);
void Create_Empty_CL(CLinklist &CL){
    CL=NULL;
}
void Insert_Head_CL(int elem,CLinklist &CL,int pos){
    int index=1;
    CLnode* p=CL;
    while(index<pos){
        p=p->next;
        index++;
    }
    CLnode* temp=(CLinklist)malloc(sizeof(CLnode));
    temp->key=elem;
    temp->next=NULL;
    temp->next=p->next;
    p->next=temp;
}
void Insert_Tail_CL(int elem,CLinklist &CL,int idn){
    //int idn=1;
    CLnode* temp=(CLinklist)malloc(sizeof(CLnode));
    temp->id=idn;
    temp->key=elem;
    temp->next=NULL;
    CLnode* p=CL;
    if(CL==NULL){
        p=temp;
        CL=temp;
        p->next=CL;
    }else{
        while(p->next!=CL){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}

void Display_CL(CLinklist &CL){

```

```

CLnode* p=CL;
if(p!=NULL){
    if(p->next==CL){
        printf("(%d,%d)",p->id,p->key);
    }else{
        printf("(%d,%d)",p->id,p->key);
        printf("->");
        p=p->next;
        while(p!=CL){

            printf("(%d,%d)",p->id,p->key);
            //printf("%d",p->key);
            if(p->next!=CL){
                printf("->");
            }
            p=p->next;
        }
        printf("\n");
    }else{
        exit(0);
    }
}

CLnode* Solution_Joseph(int m,int numperson,int* keyperson,CLinklist &CL){
    CLinklist res;
    Create_Empty_CL(res);
    Create_Empty_CL(CL);

    CLnode* op=res;
    int index;
    int id=1;
    // CL->id=id;
    // CL->key=keyperson[0];
    for(index=0;index<numperson;index++){
        Insert_Tail_CL(keyperson[index],CL,id+index);
    }
    CLnode* p=CL;
    CLnode* pre=CL;
    int times=1;
    //p=p->next;
    // int flag=0;
    while(pre->next!=p){

```

```

        pre=pre->next;
    }

    while(CL!=NULL){
        if(CL->next==CL){
            Insert_Tail_CL(CL->id,res,id++);
            break;
        }
        if(times==m){

            CLnode* temp=p;
            if(pre->next==CL){

                CL=CL->next;
                p=CL;
                pre->next=temp->next;
                temp->next=NULL;
                m=temp->key;
                times=1;
                Insert_Tail_CL(temp->id,res,id++);
                free(temp);
            }else{
                p=p->next;
                pre->next=temp->next;
                temp->next=NULL;
                m=temp->key;
                times=1;
                Insert_Tail_CL(temp->id,res,id++);
                free(temp);
            }
        }else{
            times++;
            p=p->next;
            while(pre->next!=p){

                pre=pre->next;
            }
        }
    }
    return res;
}

int main(){
    int m=20;
    int numperson=7;
    int a[7]={3,1,7,2,4,8,4};
    CLnode* l;
    int* p=&a[0];

```

```
CLnode* res=Solution_Joseph(m,numperson,p,l);
Display_CL(res);
system("pause");
}
```

### 三、实验环境

实验环境：

1. 操作系统：

windows10

2. 调试软件名称及版本号

Visual Studio Code February 2022 (version 1.65)

3. 编程语言及版本号

C++/C

4. 上机地点

班级未集中进行实验课

5. 机器台号

华硕飞行堡垒 8

### 四、实验过程与分析

## 实验 1.1

### 1. 算法分析

主函数：

```
int main(){
    DCLinkedlist a;
    DCLinkedlist b;
    char caa[]={-1111,1111};
    char cba[]={1111,1111};
    char* ca=&caa[0];
    char* cb=&cba[0];
    Init_list(a,ca);
    Init_list(b,cb);
    add_list(a,b);
    system("pause");
}
```

主要函数：

**int Init\_list(DCLinkedlist &list,char\* input);**: 根据字符串初始化对应的双向循环链表  
**void Display(const DCLinkedlist &list);**打印对应的双向循环链表信息  
**void sub\_list(DCLinkedlist &a,DCLinkedlist &b);**对双向循环链表 a, b 进行减法操作。(实验中未要求，但是会涉及到减法的需求)  
**void add\_list(DCLinkedlist &a,DCLinkedlist &b);**双向循环链表 a, b 进行加法操作  
**void run\_caculate(DCLinkedlist &a,DCLinkedlist &b);** 主逻辑运行函数，根据链表数值的正负来分别进入对应的函数操作，进行加法运算

### (1)Init\_list

```
int Init_list(DCLinkedlist &list,char* input){
    list=(DCLinkedlist)malloc(sizeof(DCLNode));
    list->next= NULL;
    list->pre=NULL;
    if(list==NULL){
        return 0;
    }
    int i=0;
    DCLNode* p=list;
    if(input[i]=='-' ){
        Elemtyp e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=1;
        list->data=e;
        i++;
    }else{
        Elemtyp e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=0;
        list->data=e;
    }
    while(input[i]!='\0'){
        int index=0;
        char temp[5]="";
        while(input[i]!=',' &&input[i]!='\0'){
            temp[index++]=input[i++];
        }
        int data=atoi(temp);
        DCLinkedlist newnode=(DCLinkedlist)malloc(sizeof(DCLNode));
        newnode->next=NULL;
        newnode->pre=NULL;
        Elemtyp e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=data;
        newnode->data=e;
        p->next=newnode;
        newnode->pre=p;
        p=newnode;
        if(input[i]==',' ){
            i++;
            continue;
        }
    }
}
```

```

    if(input[i]=='\0'){
        p->next=list;
        list->pre=p;
        break;
    }

}
return 1;
}

```

此函数主要根据字符串初始化对应的双向循环链表。传入的字符串为 char\*类型的 input,如果传入的数值为负，则将链表的头节点的数据域设置为 1，否则设置为 0。然后根据如数字字符串分隔符“，”来实现对字符串数字的分割，并链接呈一个完成的链表。如果循环到字符串的最末尾，链接链表的首部和尾部构造完整的双向循环链表。

**时间复杂度:** O(n),n 为字符串长度

**空间复杂度:** O(1)

## (2)Display

```

void Display(const DCLinkedlist &list){
    DCLinkedlist p,q;
    p=list;
    q=list;
    p=p->next;
    q=q->next;
    int sum=0;
    while(p!=list){
        sum=sum+*(int*)p->data.type;
        p=p->next;
    }
    if(sum==0){
        printf("%d",sum);
        return;
    }
    if(*(int*)list->data.type==1){
        printf("-");
    }
    while(q->next!=list){
        if(*(int*)q->data.type==0){
            printf("0000,");
        }else{
            if(list->next!=q){
                int times=0;
                if(0<*(int*)q->data.type&&*(int*)q->data.type<10){
                    times=3;
                }
                if(9<*(int*)q->data.type&&*(int*)q->data.type<100){

```

```

        times=2;
    }
    if(99<*(int*)q->data.type&&*(int*)q->data.type<1000){
        times=1;
    }
    for(int index=0;index<times;index++){
        printf("0");
    }
    printf("%d",*(int*)q->data.type);
    printf(",");
}
q=q->next;
}
if(*(int*)q->data.type==0){
    printf("0000");
}else{
    int times=0;
    if(0<*(int*)q->data.type&&*(int*)q->data.type<10){
        times=3;
    }
    if(9<*(int*)q->data.type&&*(int*)q->data.type<100){
        times=2;
    }
    if(99<*(int*)q->data.type&&*(int*)q->data.type<1000){
        times=1;
    }
    for(int index=0;index<times;index++){
        printf("0");
    }
    printf("%d",*(int*)q->data.type);
}
}

}

```

此函数主要打印对应的双向循环链表信息。根据传入的链表，首先判断数据域上各个数值是否均为 0，若为 0 则打印 0 结束函数。否则，判断符号位是否为 1（前面已经约定好 1 代表负数，0 代表正数），判断是否打印负号。然后循环双向循环链表，打印每个结点的数值。最终为结果。

**时间复杂度：O(n),n 为字符串长度**

**空间复杂度：O(1)**

### (3) add\_list

```

void add_list(DCLinkedlist &a,DCLinkedlist &b){
    DCLinkedlist enda=a->pre;

```

```

DCLinkedlist endb=b->pre;
int carry=0;
if(length(a)>=length(b)){
    while(enda!=a&&endb!=b){
        *(int*)enda->data.type=*(int*)enda->data.type+*(int*)endb->data.type+carry;
        carry=0;
        if(*(int*)enda->data.type>=10000){
            *(int*)enda->data.type=*(int*)enda->data.type-10000;
            carry=1;
        }
        enda=enda->pre;
        endb=endb->pre;
    }
    while(enda!=a){
        *(int*)enda->data.type=*(int*)enda->data.type+carry;
        carry=0;
        if(*(int*)enda->data.type>=10000){
            *(int*)enda->data.type=*(int*)enda->data.type-10000;
            carry=1;
        }
        enda=enda->pre;
    }
    if(carry==1){
        DCLinkedlist newnode=(DCLinkedlist)malloc(sizeof(DCLNode));
        newnode->next=NULL;
        newnode->pre=NULL;
        Elemtyp e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=carry;
        newnode->data=e;

        newnode->next=a->next;
        a->next->pre=newnode;
        newnode->pre=a;
        a->next=newnode;
    }
    Display(a);
}else{
    while(enda!=a&&endb!=b){
        *(int*)endb->data.type=*(int*)enda->data.type+*(int*)endb->data.type+carry;
        carry=0;
        if(*(int*)endb->data.type>=10000){
            *(int*)endb->data.type=*(int*)endb->data.type-10000;
            carry=1;
        }
    }
}

```

```

        }
        enda=enda->pre;
        endb=endb->pre;
    }
    while(endb!=b){
        *(int*)endb->data.type=*(int*)endb->data.type+carry;
        carry=0;
        if(*(int*)endb->data.type>=10000){
            *(int*)endb->data.type=*(int*)endb->data.type-10000;
            carry=1;
        }
        endb=endb->pre;
    }
    if(carry==1){
        DCLinkedlist newnode=(DCLinkedlist)malloc(sizeof(DCLNode));
        newnode->next=NULL;
        newnode->pre=NULL;
        Elemtyp e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=carry;
        newnode->data=e;
        newnode->next=b->next;
        b->next->pre=newnode;
        newnode->pre=b;
        b->next=newnode;
    }
    Display(b);
}

```

此函数主要对双向循环链表 **a**, **b** 进行加法操作。首先声明链表 a,b 的尾指针 enda,endb，其分别指向操作数的第一个万进制数。初始化进位 carry 为 0。接着判断链表 a 和 b 的长度，对长度大的链表进行最终的操作。例如如果链表 a 长度大于 b，那么就把 b 的值操作到 a 链表上，最终 Display 传入的就是 a，输出的也是链表 a 的信息。然后当尾指针 enda,endb 均没有走到对应链表头节点的条件下循环操作每个链表的结点，如果和大于 10000，则令 carry 等于 1，继续参与下轮循环。当 enda,endb 尾指针均到达符号位之后，判断 carry 是否为 1，若为 1，则需要对链表进行新增结点的操作。最终打印出对应的链表信息。

**时间复杂度：**  $O(\max(m,n))$ ,n 为链表 a 长度,m 为链表 b 长度

**空间复杂度：**  $O(1)$

#### (4) sub\_list

```

void sub_list(DCLinkedlist &a,DCLinkedlist &b){
    DCLinkedlist enda=a->pre;
    DCLinkedlist endb=b->pre;
    int borrow=0;

```

```

if(length(a)>length(b)||size_cmp(a,b)){
    while(enda!=a&&endb!=b){
        *(int*)enda->data.type=*(int*)enda->data.type-*(int*)endb->data.type-borrow;
        borrow=0;
        if(*(int*)enda->data.type<0){
            *(int*)enda->data.type=*(int*)enda->data.type+10000;
            borrow=1;
        }
        enda=enda->pre;
        endb=endb->pre;
    }
    while(enda!=a){
        *(int*)enda->data.type=*(int*)enda->data.type-borrow;
        borrow=0;
        if(*(int*)enda->data.type<0){
            *(int*)enda->data.type=*(int*)enda->data.type+10000;
            borrow=1;
        }
        enda=enda->pre;
    }

    while(*(int*)a->next->data.type==0&&a->next->next!=a){
        a->next=a->next->next;
        a->next->pre=a;
    }
    Display(a);
}
if(length(a)<length(b)||size_cmp(b,a)){
    while(enda!=a&&endb!=b){
        *(int*)endb->data.type=*(int*)endb->data.type-*(int*)enda->data.type-borrow;
        borrow=0;
        if(*(int*)endb->data.type<0){
            *(int*)endb->data.type=*(int*)endb->data.type+10000;
            borrow=1;
        }
        enda=enda->pre;
        endb=endb->pre;
    }
    while(endb!=b){
        *(int*)endb->data.type=*(int*)endb->data.type-borrow;
        borrow=0;
        if(*(int*)endb->data.type<0){
            *(int*)endb->data.type=*(int*)endb->data.type+10000;
            borrow=1;
        }
    }
}

```

```

        }
        endb=endb->pre;
    }

    while(*(int*)b->next->data.type==0&&b->next->next!=b){
        b->next=b->next->next;
        b->next->pre=b;
    }
    Display(b);
}
}

```

此函数主要对双向循环链表 **a, b** 进行减法操作。首先声明链表 a,b 的尾指针 enda,endb，其分别指向操作数的第一个万进制数。初始化借位 borrow 为 0。接着判断链表 a 和 b 的长度，a 和 b 绝对值的大小，对长度大的链表或者绝对值大的链表进行最终的操作，同时最终结果的符号和被操纵链表的符号位相同。然后就是循环操作。最终判断需要打印的链表符号位后的数据域是否为 0，删除为 0 的数据域直到仅剩一个结点或者下一节点数据域不为 0。

**时间复杂度：O(MAX(m,n)),n 为链表 a 长度,m 为链表 b 长度**

**空间复杂度：O(1)**

## (5) run\_caculate

```

void run_caculate(DCLinkedlist &a,DCLinkedlist &b){
    if(*(int*)a->data.type==*(int*)b->data.type){
        add_list(a,b);
    }else{
        sub_list(a,b);
    }
}

```

此函数是主逻辑运行函数，根据链表数值的正负来分别进入对应的函数操作，进行加法运算。对于数据的操作逻辑已经在对应的 add\_list 和 sub\_list 方法里声明。因此这里只需要判断操作数的符号是否相同即可。若相同则调用 add——list 方法，否则调用 sub\_list 方法。

**时间复杂度：O(1)**

**空间复杂度：O(1)**

**算法设计巧妙之处：**对于实验 1.1，整体算法的复杂度均控制在线性时间内。代码逻辑清晰，操作简单，容易理解。并且自定义了一个新的数据类型 Elemtpe，该结构体可以实现存储任意基本数据类型的功能，使得代码具有一定的普适性，提高了代码的复用性。将加法巧妙的转换为加法和减法，分类讨论，简化了问题的处理。依据题意要求，实现了双向循环链表作为本实验的基本存储结构，合理利用了头节点的结构存储符号标志位。符合题意要求的同时保留了实验的创新性。

## 实验 1.2

### 1. 算法分析

主函数

```
int main(){
    int m=20;
    int numperson=7;
    int a[7]={3,1,7,2,4,8,4};
    CLnode* l;
    int* p=&a[0];
    CLnode* res=Solution_Joseph(m,numperson,p,l);
    Display_CL(res);
    system("pause");
}
```

主要函数

```
CLnode* Solution_Joseph(int m,int numperson,int* keyperson,CLinklist &CL);
```

```
CLnode* Solution_Joseph(int m,int numperson,int* keyperson,CLinklist &CL){
```

```
    CLinklist res;
    Create_Empty_CL(res);
    Create_Empty_CL(CL);

    CLnode* op=res;
    int index;
    int id=1;
    for(index=0;index<numperson;index++){
        Insert_Tail_CL(keyperson[index],CL,id+index);
    }
    CLnode* p=CL;
    CLnode* pre=CL;
    int times=1;
    while(pre->next!=p){
        pre=pre->next;
    }
    while(CL!=NULL){
        if(CL->next==CL){
            Insert_Tail_CL(CL->id,res,id++);
            break;
        }
        if(times==m){

            CLnode* temp=p;
            if(pre->next==CL){
                CL=CL->next;
                p=CL;
            }
        }
    }
}
```

```

    pre->next=temp->next;
    temp->next=NULL;
    m=temp->key;
    times=1;
    Insert_Tail_CL(temp->id,res,id++);
    free(temp);
}else{
    p=p->next;
    pre->next=temp->next;
    temp->next=NULL;
    m=temp->key;
    times=1;
    Insert_Tail_CL(temp->id,res,id++);
    free(temp);
}
}else{
    times++;
    p=p->next;
    while(pre->next!=p){
        pre=pre->next;
    }
}
}
return res;
}

```

此函数为 Joseph 问题的主逻辑函数。函数传入参数：m 为初始的目标值，numperson 为参与游戏人数，keyperson 为每个参与人的 key 值，CL 为传入的操作链表。首先初始化操作链表信息，初始化就绪链表为空。初始化循环次数为 0。当操作链表不为空时，进入循环，不断找出对于结点的 key 值等于循环次数的结点，将其插入到就绪队列中。返回就绪链表头结点的地址。便于打印操作。

**时间复杂度：** $O(n*m)$ ，因为每次都要报数报  $m$  次，不考虑具体是什么数据结构实现存储， $n$  个人需要报  $n - 1$  次数，那么问题规模是  $O(n * m)$ 。

**空间复杂度：** $O(n)$ ，在实验中有多少个人，我们就声明了若干长度为  $n$  的数据结构，包括  $n$  个人的  $m$  值信息和链接他们的链表。

**算法设计巧妙之处：**对于实验 1.2，整体算法的复杂度均控制在线性时间内。代码逻辑清晰，操作简单，容易理解。代码具有一定的普适性和复用性。依据题意要求，实现了没有头节点的单向循环链表作为本实验的基本存储结构，保留了实验的创新性。

## 2. 调试过程

## 实验 1.1:

1. 读取字符的数组未初始化，会出现乱码错位。

修改前：

问题代码：

```
while(input[i]!='\0'){
    int index=0;
    //char temp[5]="";
    char temp[5];
    while(input[i]!=','&&input[i]!='\0'){
        temp[index++]=input[i++];
    }
}
```

运行数据：

```
int main(){
    DCLinkedlist a;
    DCLinkedlist b;
    char caa[]={1,4567};
    char cba[]={2,1111};
    char* ca=&caa[0];
    char* cb=&cba[0];
    Init_list(a,ca);
    Init_list(b,cb);
    add_list(a,b);
    system("pause");
}
```

运行结果：

```
c:\VS\code\c and c++\datastructure\lab1.4.exe
2568, 5678请按任意键继续. . .
```

答案明显错误。

修改后：

正确代码：

```
while(input[i]!='\0'){
    int index=0;
    char temp[5]="";
    //char temp[5];
    while(input[i]!=','&&input[i]!='\0'){
        temp[index++]=input[i++];
    }
}
```

运行数据：

```
int main(){
    DCList a;
    DCList b;
    char caa[]="1,4567";
    char cba[]="2,1111";
    char* ca=&caa[0];
    char* cb=&cba[0];
    Init_list(a,ca);
    Init_list(b,cb);
    add_list(a,b);
    system("pause");
}
```

运行结果：

```
c:\VS\code\c and c++\datastructure\lab1.4.exe
3, 5678请按任意键继续. . .
```

结论：所有变量的声明都需要提前初始化。

2. 数据域小于 1000 时，0 不显示。

修改前：

问题代码：

```
while(q->next!=list){
    if(*(int*)q->data.type==0){
        printf("0000,");
    }else{
        // if(list->next!=q){
        //     int times=0;
        //     if(0<*(int*)q->data.type&&*(int*)q->data.type<10){
        //         times=3;
        //     }
        //     if(9<*(int*)q->data.type&&*(int*)q->data.type<100){
        //         times=2;
        //     }
        //     if(99<*(int*)q->data.type&&*(int*)q->data.type<1000){
        //         times=1;
        //     }
        //     for(int index=0;index<times;index++){
        //         printf("0");
        //     }
        // }
        printf("%d,*(int*)q->data.type);
        printf(",");
    }
    q=q->next;
}
```

运行数据：

```

int main(){
    DCList a;
    DCList b;
    char caa[]={34,5544,5367};
    char cba[]={-5,5443,5333};
    char* ca=&caa[0];
    char* cb=&cba[0];
    Init_list(a,ca);
    Init_list(b,cb);
    run_caculate(a,b);
    system("pause");
}

```

运行结果：

```

c:\VS\code\c and c++\datastructure\lab1.4.exe
29, 101, 0034请按任意键继续. . .

```

答案明显错误。

修改后：

正确代码：

```

while(q->next!=list){
    if(*(int*)q->data.type==0){
        printf("0000,");
    }else{
        if(list->next!=q){
            int times=0;
            if(0<*(int*)q->data.type&&*(int*)q->data.type<10){
                times=3;
            }
            if(9<*(int*)q->data.type&&*(int*)q->data.type<100){
                times=2;
            }
            if(99<*(int*)q->data.type&&*(int*)q->data.type<1000){
                times=1;
            }
            for(int index=0;index<times;index++){
                printf("0");
            }
        }
        printf("%d",*(int*)q->data.type);
        printf(",");
    }
    q=q->next;
}

```

运行数据：

```
int main(){
    DCList a;
    DCList b;
    char caa[] = "34,5544,5367";
    char cba[] = "-5,5443,5333";
    char* ca = &caa[0];
    char* cb = &cba[0];
    Init_list(a, ca);
    Init_list(b, cb);
    run_caculate(a, b);
    system("pause");
}
```

运行结果：

```
[c:\VS\code\c and c++\datastructure\lab1.4.exe
29,0101,0034请按任意键继续. . . ]
```

对于输出 0 的逻辑需要整理清楚。

## 实验 1.2：

实验中遇到了若干大问题，例如在编写程序之前并未注意到题目提示里面所说的不需要使用头节点。但是自己在第一次做的时候还是使用了头节点，后续 debug 测试过程中发现使用头节点会存在很多特殊情况无法讨论，特别是 vscode 报段异常的错误，十分棘手。最终未采用头节点，问题顺利解决！

### 五、实验结果总结

回答以下问题：

(1) 你的测试充分吗？为什么？你是怎样考虑的？

答：对于实验 1.1，我认为我的测试是比较充分的。实验过程见下图：

测试一：

```
char caa[] = "0";
char cba[] = "0";
```

```
[c:\VS\code\c and c++\datastructure\lab1.4.exe
0请按任意键继续. . . ]
```

测试二：

```
char caa[] = "-2345,6789";
char cba[] = "-7654,3211";
```

```
[c:\VS\code\c and c++\datastructure\lab1.4.exe
-1,0000,0000请按任意键继续. . . ]
```

满足所有的测试数据，实验达到预期。

对于实验 1.2，我认为我的测试是比较充分的。实验过程见下图：

测试一：

```
int main(){
    int m=20;
    int numperson=7;
    int a[7]={3,1,7,2,4,8,4};
    CLnode* l;
    int* p=&a[0];
    CLnode* res=Solution_Joseph(m,numperson,p,l);
    Display_CL(res);
    system("pause");
}
```

预期结果：

【测试数据】

m 的初值为 20, n=7,7 个人的密码依次为：3、1、7、2、4、8、4，首先 m 值为 6（正确的出列顺序应为 6、1、4、7、2、3、5）。

6, 1, 4, 7, 2, 3, 5

实验结果：

```
c:\VS\code\c and c++\datastructure\lab1.1.exe
(1, 6)->(2, 1)->(3, 4)->(4, 7)->(5, 2)->(6, 3)->(7, 5)
请按任意键继续. . .
```

输出的结果是就绪链表的所有信息。形式(a,b)中，a 表示链表的标识符 ID，b 表示位于原来操作链表的顺序值。

(2) 你的存储结构的选取是不是很适合这个应用？为什么？

答：对于**实验 1.1**，我所选择的存储结构很适合这个应用。依据题意，实验要求我们使用双向循环链表来实现。双向链表的每个数据结点中都有两个指针，分别指向直接后继和直接前驱。所以，从双向链表中的任意一个结点开始，都可以很方便地访问它的前驱结点和后继结点。

对于**实验 1.2**，我所选择的存储结构很适合这个应用。依据题意，实验要求我们使用不带头节点的链表来实现。选取使用链表最明显的好处就是，常规数组排列关联项目的方式可能不同于这些数据项目在记忆体或磁盘上顺序，数据的存取往往要在不同的排列顺序中转换。链表允许插入和移除表上任意位置上的节点，但是不允许随机存取。

(3) 用一段简短的代码及说明论述你的应用中有关插入和删除元素是如何做的？

答：对于**实验 1.1**，插入元素的代码：

```
if(carry==1){  
    DCLinkedlist newnode=(DCLinkedlist)malloc(sizeof(DCLNode));  
    newnode->next=NULL;  
    newnode->pre=NULL;  
    Elemtyp e;  
    e.type=(int*)malloc(sizeof(int));  
    *(int*)e.type=carry;  
    newnode->data=e;  
  
    newnode->next=b->next;  
    b->next->pre=newnode;  
    newnode->pre=b;  
    b->next=newnode;  
}
```

如果进位等于 1，那么说明原来的链表所装填的数据已经满了，需要一个新的结点来装填额外的变量。在代码中，b 指向头节点也就是符号位。申请一个新的空间被 newnode 指向，令其前驱和后继均为 NULL，将进位赋值给 newnode。将 newnode 的 next 指针指向头结点的下一个节点，同时头结点的下一个节点的 pre 指针指向 newnode，然后 newnode 的 pre 指针指向头节点 b，b 的 next 指针指向 newnode。完成 newnode 的头部插入。

删除元素的代码：

```
while(*(int*)a->next->data.type==0&&a->next->next!=a){  
    a->next=a->next->next;  
    a->next->pre=a;  
}
```

如果进行操作完之后，发现出现了-0000, 0000, 0010 形式的结果，则需要进行删除元素的操作。在代码中，a 指向头节点也就是符号位。当头节点的下一个节点的数据域为 0 并且头节点的下二个结点不等于头结点时，不断地删除头结点的下一个节点：头结点的 next 指针指向头结点的下两个结点，同时头结点的 pre 指针指向头节点。即完成链表的头部删除。

对于**实验 1.2**，插入代码：

```

void Insert_Tail_CL(int elem, CLinklist &CL, int idn){
    //int idn=1;
    CLnode* temp=(CLinklist)malloc(sizeof(CLnode));
    temp->id=idn;
    temp->key=elem;
    temp->next=NULL;
    CLnode* p=CL;
    if(CL==NULL){
        p=temp;
        CL=temp;
        p->next=CL;
    }else{
        while(p->next!=CL){
            p=p->next;
        }
        temp->next=p->next;
        p->next=temp;
    }
}

```

首先循环遍历到链表尾部，进行尾部插入的缝合即可。

删除代码：

```

p=p->next;
pre->next=temp->next;
temp->next=NULL;
m=temp->key;
times=1;
Insert_Tail_CL(temp->id,res,id++);
free(temp);

```

构造 temp 结点为待删除的结点，链接 temp 前后的结点，free 掉 temp 结点即可。

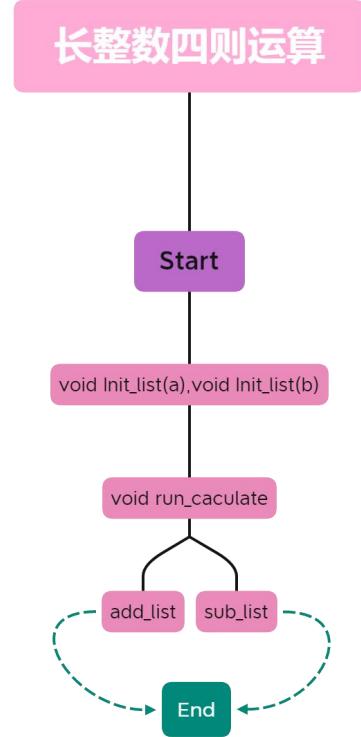
(4) 在你的应用中是否用到了头结点？你觉得使用头结点为你带来方便了吗？

答：对于**实验 1.1**，使用到了头节点。头节点在本实验中充当存储符号位的功能，带来了很大的方便。

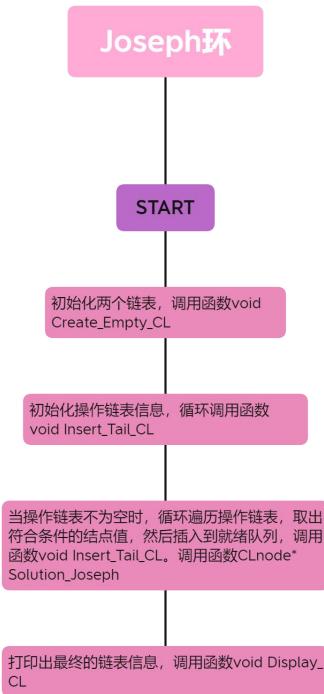
对于**实验 1.2**，并没有使用头节点。本实验需要不断删除结点元素，头节点在本实验中并不具有明显的功能与意义，反而会增加算法的复杂度，影响效率。

(5) 源程序的大致的执行过程是怎样的？

答：对于**实验 1.1**，源程序的执行流程图如下：



对于实验 1.2，源程序的执行流程图如下：



## 六、附录

1. 其他的解决方案或设想，或对实验方案的意见：

(1) 长整数四则运算题目要求仅仅实现了加法，其他三则运算（减法，乘法，除法）未要求，可以适当增加难度。

2. 实验参考的资料和网址：

(1) [https://blog.csdn.net/qq\\_44310495/article/details/109395413?ops\\_request\\_misc=%257B%2522request%2522%25Fid%2522%253A%2522165312667516781435483636%2522%252C%2522scm%2522%253A%252220140713.130102334.%2522%257D&request\\_id=165312667516781435483636&biz\\_id=0&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~top\\_positive~default-1-109395413-null-null.142~v10~control,157~v4~control&utm\\_term=%E9%95%BF%E6%95%B4%E6%95%B0%E5%9B%9B%E5%88%99%E8%BF%90%E7%AE%97&spm=1018.2226.3001.4187](https://blog.csdn.net/qq_44310495/article/details/109395413?ops_request_misc=%257B%2522request%2522%25Fid%2522%253A%2522165312667516781435483636%2522%252C%2522scm%2522%253A%252220140713.130102334.%2522%257D&request_id=165312667516781435483636&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-109395413-null-null.142~v10~control,157~v4~control&utm_term=%E9%95%BF%E6%95%B4%E6%95%B0%E5%9B%9B%E5%88%99%E8%BF%90%E7%AE%97&spm=1018.2226.3001.4187)

(2) [https://blog.csdn.net/yezhuandroid/article/details/78924402?ops\\_request\\_misc=&request\\_id=&biz\\_id=102&utm\\_term=Joseph%E7%8E%AF%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~sobaiduweb~default-1-78924402.142~v10~pc\\_search\\_result\\_control\\_group,157~v12~control&spm=1018.2226.3001.4187](https://blog.csdn.net/yezhuandroid/article/details/78924402?ops_request_misc=&request_id=&biz_id=102&utm_term=Joseph%E7%8E%AF%E6%97%B6%E9%97%B4%E5%A4%8D%E6%9D%82%E5%BA%A6&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-78924402.142~v10~pc_search_result_control_group,157~v12~control&spm=1018.2226.3001.4187)

教师评语或评价表格：（任课教师可根据实际情况，做适当调整）

评语及评价表格的字体颜色为红色

评价表格：

评价内容	具体要求	分值	得分
平时表现	课程设计过程中，无缺勤现象，态度积极，具有严谨的学习态度和认真、踏实、一丝不苟的科学作风。	20	
报告质量	实验报告格式规范，符合要求；报告内容充实、正确，实验目的归纳合理到位。	30	
实验内容	能够按实验要求合理设计并开发出程序，功能完整性 强，原理及实验结果分析准确，归纳总结充分。	50	
总 分			

# 实验报告

课程名称：数据结构与算法分析 班级：软信 2001 实验成绩：

实验名称：栈、队列、字符串和 学号：20207130 批阅教师签字：  
数组

实验编号：实验二 姓名：赖骏鸿 实验日期：2020 年 5 月 19 日

指导教师：马毅 组号：暂无 实验时间：18 时 30 分—22 时 15 分

## 一、实验目的

- (1) 掌握栈、队列、串和数组的抽象数据类型的特征。
- (2) 掌握栈、队列、串和数组的抽象数据类型在计算机中的实现方法。
- (3) 学会使用栈、队列来解决一些实际的应用问题。

## 二、实验内容与实验步骤

### 1. 实验内容

迷宫问题（难度系数为 4，记作实验 2.1），算术表达式求值问题（难度系数为 5，记作实验 2.2）

#### 实验 2.1：

##### [问题描述]

以一个  $m \times n$  的长方阵表示迷宫，0 和 1 分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的额迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

##### [基本要求]

首先实现一个以链表作存储结构的栈类型，然后编写一个求解迷宫问题的非递归算法。求得的通路以三元组  $(i, j, d)$  的形式输出，其中： $(i, j)$  指示迷宫中的， $d$  表示走到下一个坐标的方向。如：对于下列数据的迷宫，输出的一条通路为：  $(1, 1, 1), (1, 2, 2), (2, 2, 2), (3, 2, 3), (3, 1, 2) \dots$

##### [测试数据]

迷宫的测试数据如下：左上角  $(1, 1)$  为入口，右下角  $(8, 9)$  为出口

```
0, 0, 1, 0, 0, 0, 1, 0,  
0, 0, 1, 0, 0, 0, 1, 0,  
0, 0, 0, 0, 1, 1, 0, 1,  
0, 1, 1, 1, 0, 0, 1, 0,  
0, 0, 0, 1, 0, 0, 0, 0,  
0, 1, 0, 0, 0, 1, 0, 1,  
0, 1, 1, 1, 0, 0, 1,  
1, 1, 0, 0, 0, 1, 0, 1,  
1, 1, 0, 0, 0, 0, 0, 0
```

##### [实现提示]

计算机解迷宫问题常用的是“穷举求解”，即从入口触发，顺着一个方向进行探索，若能走通，则继续前进；否则沿着回路退回，换一个方向继续探索，直至出口位置，求得一条

通路，假设所有可能的通路都探索到而未能到达出口，则所设定的迷宫没有通路。

可以二维数组存储迷宫数据，通常设定入口点的下标（1, 1）出口点的下标为（n,n），为处理方便起见，可在迷宫的四周加一圈障碍。对于迷宫中任一位置，均可约定有东、南、西、北四个方向可通。

## 实验 1.2:

### [问题描述]

表达式计算是实现程序设计语言的基本问题之一。也是栈的应用的一个典型例子。设计一个程序，演示用算符优先法或转换成后缀表达式方法对算术表达式进行求值的过程。

### [基本要求]

以字符序列的形式从终端输入语法正确的、不含变量的整数表达式。利用教科书表 3.1 给出的算符优先关系，或课件中给出的中缀表达式向后缀表达式转换的方法，实现对算术四则混合运算表达式的求值，并仿照教科书和课件中的例子演示求值中运算符栈、运算数栈、输入字符和主要操作变化的过程。

### [测试数据]

3 \* (7 - 2)

8; 1 + 2 + 3 + 4; 88 - 1 \* 5; 1024 / 4 \* 8; (6 + 2 \* (3 + 6 \* (6 + 6)))

## 实验 1.1:

### 2. 数据结构及函数说明

#### (1) 基本数据

```

#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
#include<limits.h>
const int col=8;//提前定义好的列数
const int row=9;//提前定义好的行数
const int ylen=col+2;//定义横坐标
const int xlen=row+2;
int maze[row][col]={
    0, 0, 1, 0, 0, 0, 1, 0,
    0, 0, 1, 0, 0, 0, 1, 0,
    0, 0, 0, 0, 1, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1, 0,
    0, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 1, 0, 1,
    0, 1, 1, 1, 1, 0, 0, 1,
    1, 1, 0, 0, 0, 1, 0, 1,
    1, 1, 0, 0, 0, 0, 0, 0
};//定义好的地图
int new_maze[row+2][col+2]={0};//新地图，增加了墙
int angel[]={3,2,1,0};//定义方向优先级，右下左上
int visit[xlen][ylen]={0};//是否访问的二维数组标志位

```

## (2) 自定义结构体

```

typedef struct SElmtype{
    void* type;
}Elmtype;//自定义模板数据类型

typedef struct LNode{
    Elmtype data;
    struct LNode* next;
}LNode,*Linkstack;//自定义栈

```

## (3) 函数说明

```

int Find_angel(int x,int y);//寻找移动角度
void Init_stack(LinkStack &stack);//初始化栈
int Isempty_stack(LinkStack &stack);//判断栈是否为空
int push_stack(LinkStack &stack,SElmtype e);//入栈
SElmtype pop_stack(LinkStack &stack);//出栈
SElmtype top_stack(LinkStack &stack);//取出栈顶元素
void Get_maze();//初始化新地图
void Print_maze();//打印新地图
void Print_stack(LinkStack &stack);//打印栈信息
void Print_order(LinkStack &x,LinkStack &y,LinkStack &angel);//打印路线信息

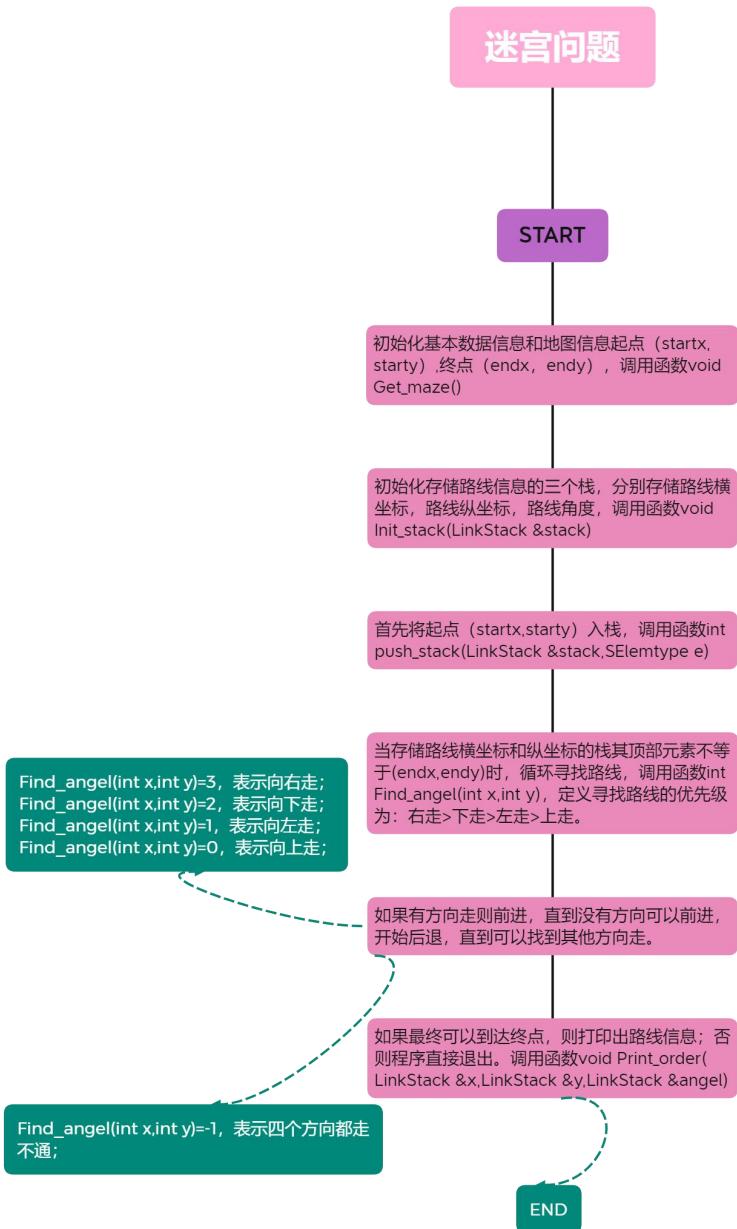
```

函数	说明
int Find_angel(int x,int y);	寻找移动角度
void Init_stack(LinkStack &stack);	初始化栈
int Isemptry_stack(LinkStack &stack);	判断栈是否为空
int push_stack(LinkStack &stack,SElemtype e);	入栈
SElemtype pop_stack(LinkStack &stack);	出栈
SElemtype top_stack(LinkStack &stack);	取出栈顶元素
void Get_maze();	初始化新地图
void Print_maze();	打印新地图
void Print_stack(LinkStack &stack);	打印栈信息
void Print_order(LinkStack &x,LinkStack &y,LinkStack &angel);	打印路线信息

### 3. 实验思路

首先初始化地图，给系统默认的地图四周加上墙，表示行走的路线不能越过地图边界。初始化3个栈，分别存储找到出口路线的横坐标x，纵坐标y，移动角度angel。首先将入口(x0,y0)入栈，定义地图路线的终点(endx,endy)。开始循环，当存储横坐标栈的栈顶或者存储纵坐标的栈顶分别不等于endx或者endy时，寻找路线。定义寻找路线的优先级为：右走>下走>左走>上走。如果有方向走则前进，直到没有方向可以前进，开始后退，直到可以找到其他方向走。如果最终可以到达终点，则打印出路线信息；否则程序直接退出。

### 4. 实验流程图



## 5. 源代码

```

#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
#include<limits.h>

const int col=8;//提前定义好的列数
const int row=9;//提前定义好的行数
const int ylen=col+2;//定义横坐标
const int xlen=row+2;
int maze[row][col]={
    0, 0, 1, 0, 0, 0, 1, 0,
    0, 0, 1, 0, 0, 0, 1, 0,
    0, 0, 0, 0, 1, 1, 0, 1,
}

```

```

    0, 1, 1, 1, 0, 0, 1, 0,
    0, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 1, 0, 1,
    0, 1, 1, 1, 0, 0, 1,
    1, 1, 0, 0, 0, 1, 0, 1,
    1, 1, 0, 0, 0, 0, 0, 0
};//定义好的地图

int new_maze[row+2][col+2]={0};//新地图，增加了墙
int angel[]={3,2,1,0};//定义方向优先级，右下左上
int visit[xlen][ylen]={0};//是否访问的二维数组标志位

typedef struct SElemtype{
    void* type;
}SElemtype;//自定义模板数据类型

typedef struct LNode{
    ELEMTYPE data;
    struct LNode* next;
}LNode,*LinkStack;//自定义栈

int Find_angel(int x,int y); //寻找移动角度
void Init_stack(LinkStack &stack); //初始化栈
int Isempty_stack(LinkStack &stack); //判断栈是否为空
int push_stack(LinkStack &stack,SElemtype e); //入栈
SElemtype pop_stack(LinkStack &stack); //出栈
SElemtype top_stack(LinkStack &stack); //取出栈顶元素
void Get_maze(); //初始化新地图
void Print_maze(); //打印新地图
void Print_stack(LinkStack &stack); //打印栈信息
void Print_order(LinkStack &x,LinkStack &y,LinkStack &angel); //打印路线信息
int Find_angel(int x,int y){
    if(new_maze[x][y+1]==0&&visit[x][y+1]==0){
        return 3;
    }
    if(new_maze[x+1][y]==0&&visit[x+1][y]==0){
        return 2;
    }
    if(new_maze[x][y-1]==0&&visit[x][y-1]==0){
        return 1;
    }
    if(new_maze[x-1][y]==0&&visit[x-1][y]==0){
        return 0;
    }
    return -1;
}//寻找移动角度

void Init_stack(LinkStack &stack){

```

```
stack=(LinkStack)malloc(sizeof(LNode));
stack->next=NULL;
}//初始化栈
int Isempty_stack(LinkStack &stack){
    return (stack->next==NULL?1:0);
}//判断栈是否为空
int push_stack(LinkStack &stack,SElemtype e){
    LNode *temp=(LinkStack)malloc(sizeof(LNode));
    if(!temp){
        return 0;
    }
    temp->next=NULL;
    temp->data=e;
    temp->next=stack->next;
    stack->next=temp;
    return 1;
}//入栈
SElemtype pop_stack(LinkStack &stack){
    if(Isempty_stack(stack)){
        // Elemtpe e;
        // e.type=(int*)malloc(sizeof(int));
        // *(int*)e.type=-1;
        // return e;
        exit(-1);
    }else{
        LNode* temp=stack->next;
        SElemtype e;
        stack->next=temp->next;
        temp->next=NULL;
        e=temp->data;
        free(temp);
        return e;
    }
}//出栈
SElemtype top_stack(LinkStack &stack){
    if(Isempty_stack(stack)){
        // Elemtpe e;
        // e.type=(int*)malloc(sizeof(int));
        // *(int*)e.type=-1;
        // return e;
        exit(-1);
    }
    return stack->next->data;
}//取出栈顶元素
```

```
void Get_maze(){
    for(int i=0;i<col+2;i++){
        for(int j=0;j<row+2;j++){
            if(i==0||i==col+1){
                new_maze[j][i]=1;
                continue;
            }else if(j==0||j==row+1){
                new_maze[j][i]=1;
                continue;
            }else{
                new_maze[j][i]=maze[j-1][i-1];
                continue;
            }
        }
    }
}
```

```
//初始化新地图
void Print_maze(){
    for(int i=0;i<xlen;i++){
        for(int j=0;j<ylen;j++){
            printf("%d",new_maze[i][j]);
        }
        printf("\n");
    }
}
//打印新地图
void Print_stack(LinkStack &stack){
    while(!IsEmpty_stack(stack)){
        printf("%d",*(int*)top_stack(stack).type);
        printf(",");
        pop_stack(stack);
    }
}
//打印栈信息
void Print_order(LinkStack &x,LinkStack &y,LinkStack &angel){
    int index=1;
    while(!IsEmpty_stack(x)&&!IsEmpty_stack(y)){
        if(index==1){
            printf("(");
            printf("%d",*(int*)top_stack(x).type);
            printf(",");
            printf("%d",*(int*)top_stack(y).type);
            printf(",");
            printf("4");
            printf(")");
        }
        index++;
    }
}
```

```

        pop_stack(x);
        pop_stack(y);
    }
    printf("<--");
    printf("(");
    printf("%d",*(int*)top_stack(x).type);
    printf(",");
    printf("%d",*(int*)top_stack(y).type);
    printf(",");
    printf("%d",*(int*)top_stack(angel).type);
    printf(")");
    pop_stack(x);
    pop_stack(y);
    pop_stack(angel);
    index++;
}
}//打印寻找出口顺序
void Solution_maze(){
    LinkStack x;
    LinkStack y;
    LinkStack angel;
    Init_stack(x);
    Init_stack(y);
    Init_stack(angel);//初始化若干栈， 分别表示横坐标、纵坐标、方向坐标
    Get_maze();//初始化新地图

    int startx=1;
    int starty=1;
    int dx=startx;//横坐标
    int dy=starty;//纵坐标
    Elemtpe e1,e2;
    e1.type=(int*)malloc(sizeof(int));
    e2.type=(int*)malloc(sizeof(int));
    *(int*)e1.type=dx;
    *(int*)e2.type=dy;
    push_stack(x,e1);
    push_stack(y,e2);
    int endx=row;//定义终点的横坐标
    int endy=col;//定义终点的纵坐标
    while(*(int*)top_stack(x).type!=endx||*(int*)top_stack(y).type!=endy){//当没有达到终点时，循环寻找正确路线
        if(Find_angel(dx,dy)==3){
            dx=dx;
            dy=dy+1;
    }
}

```

```
visit[dx][dy]=1;
Elemtpe ex,ey;
ex.type=(int*)malloc(sizeof(int));
ey.type=(int*)malloc(sizeof(int));
*(int*)ex.type=dx;
*(int*)ey.type=dy;
push_stack(x,ex);
push_stack(y,ey);
Elemtpe eangel;
eangel.type=(int*)malloc(sizeof(int));
*(int*)eangel.type=3;
push_stack(angel,eangel);
if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
    break;
}
}//向右走
if(Find_angel(dx,dy)==2){
    dx=dx+1;
    dy=dy;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtpe eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=2;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
}
}//向下走
if(Find_angel(dx,dy)==1){
    dx=dx;
    dy=dy-1;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
```

```

push_stack(x,ex);
push_stack(y,ey);
Elemtype eangel;
eangel.type=(int*)malloc(sizeof(int));
*(int*)eangel.type=1;
push_stack(angel,eangel);
if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
    break;
}
}//向左走

if(Find_angel(dx,dy)==0){
    dx=dx-1;
    dy=dy;
    visit[dx][dy]=1;
    Elemtype ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtype eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=0;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
}
}//向上走

if(Find_angel(dx,dy)==-1){
    pop_stack(x);
    pop_stack(y);
    pop_stack(angel);
    dx=*(int*)top_stack(x).type;
    dy=*(int*)top_stack(y).type;
}
//走不通了，退一步
}

printf("The maze is according to the following:\n");
Print_maze();
printf("The order is according to the following:\n");
Print_order(x,y,angel);
printf("\n");

```

```

    printf("tips: In the output statement (x,y,angel), x represents the abscissa of the route, y
represents the ordinate of the route, and angel represents the angle of travel of the route. \n
Where 0 means up, 1 means left 1, 2 means down, 3 means right, and 4 means reach the end point.");

}

int main(){
    Solution_maze();
    system("pause");
}

```

## 实验 2.2:

**备注:** 对于本实验（实验 2.2，算术表达式求值），本人编写了两套程序。第一套是采用 c 语言的但是仅仅支持个位数的算术表达式求值（记作实验 2.2.1），第二套是采用 c++ 语言（使用了 c++ 的 STL 库）的可以支持多位数的算术表达式求值（记作实验 2.2.2）。后续会对两套程序均进行详细的说明。

### 2. 数据结构及函数声明

#### (1) 自定义结构体

实验 2.2.1 中自定义了链式栈：

```

typedef struct SElemtype{
    void* type;
}SElemtype;

typedef struct LNode{
    ELEMTYPE data;
    struct LNode* next;
}LNode,*LinkStack;

```

#### (2) 函数说明

实验 2.2.1：

```

void Init_stack(LinkStack &stack);
int Isempty_stack(LinkStack &stack);
int push_stack(LinkStack &stack,SElemtype e);
SElemtype pop_stack(LinkStack &stack);
SElemtype top_stack(LinkStack &stack);
int priority(char c);
int number_of_brackets(char* s,int length);
char* Infix_to_Postfix(char* infix,int length);
int char_match(char ch);
int calculate_Postfix(char* s,int length);

```

函数	说明
<code>void Init_stack(LinkStack &amp;stack);</code>	初始化栈
<code>int Isempty_stack(LinkStack &amp;stack);</code>	判断栈是否为空
<code>int push_stack(LinkStack &amp;stack,SElemtype e);</code>	入栈
<code>SElemtype pop_stack(LinkStack &amp;stack);</code>	出栈
<code>SElemtype top_stack(LinkStack &amp;stack);</code>	取出栈顶元素
<code>int priority(char c);</code>	定义字符优先级
<code>int number_of_brackets(char* s,int length);</code>	计算字符串中括号数量
<code>char* Infix_to_Postfix(char* infix,int length);</code>	中缀表达式转换为后缀表达式
<code>int char_match(char ch);</code>	定义计算符号匹配值
<code>int calculate_Postfix(char* s,int length);</code>	计算后缀表达式

实验 2.2.2:

```
int string_to_int(string str);
int judge(string a);
vector<string> init_data(string s);
int calculate_Postfix(vector<string> s);
vector<string> Infix_to_Postfix(vector<string> s);
void Print_Postfix(vector<string> s);
```

函数	说明
<code>int string_to_int(string str);</code>	将字符串转换为整数
<code>int judge(string a);</code>	定义字符的优先级
<code>vector&lt;string&gt; init_data(string s);</code>	初始化数据，分割出操作字符（包括多位 整数）
<code>int calculate_Postfix(vector&lt;string&gt; s);</code>	计算后缀表达式

---

```
vector<string> Infix_to_Postfix(vector<string> s);           中缀表达式转换为后缀表达式
void Print_Postfix(vector<string> s);                          打印后缀表达式
```

---

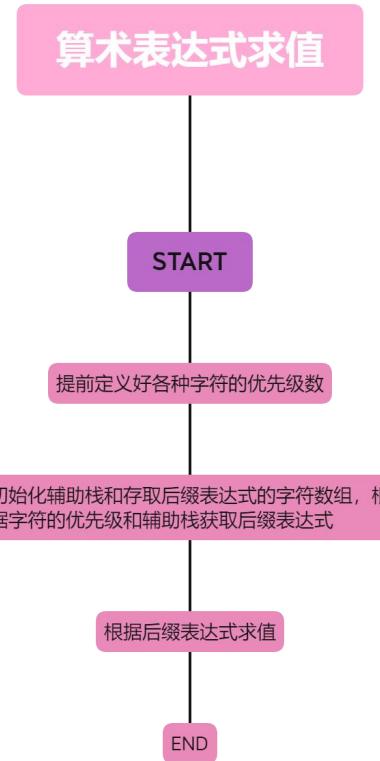
### 3.实验思路

本实验的实验思路是：首先定义字符的优先级，然后将输入的中缀表达式转换为后缀表达式，最后将后缀表达式进行计算并输出结果。其中最重要的是将输入的中缀表达式转换为后缀表达式。中缀表达式  $a + b * c + (d * e + f) * g$ ，其转换成后缀表达式则为  $a b c * + d e * f + g * +$ 。转换过程需要用到栈，具体过程如下：

- 1) 如果遇到操作数，我们就直接将其输出。
- 2) 如果遇到操作符，则我们将其放入到栈中，遇到左括号时我们也将其放入栈中。
- 3) 如果遇到一个右括号，则将栈元素弹出，将弹出的操作符输出直到遇到左括号为止。注意，左括号只弹出并不输出。
- 4) 如果遇到任何其他的操作符，如（“+”，“\*”，“（”）等，从栈中弹出元素直到遇到发现更低优先级的元素(或者栈为空)为止。弹出完这些元素后，才将遇到的操作符压入到栈中。有一点需要注意，只有在遇到" )"的情况下我们才弹出"( "，其他情况我们都不会弹出"( "。
- 5) 如果我们读到了输入的末尾，则将栈中所有元素依次弹出。

然后是后缀表达式的计算，思路是如果遇到数字就直接将其入栈，如果遇到符号，则取出栈顶两个元素进行运算，然后在入栈。如此循环直到到达表达式末尾，最后输出栈顶元素就是结果。

### 4.实验流程图



## 5.源代码

### 实验 2.2.1 源代码:

```

#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
#include<limits.h>
#include<string.h>

typedef struct SElemtype{
    void* type;
}SElemtype;
typedef struct LNode{
    ELEMTYPE data;
    struct LNode* next;
}LNode,*LinkStack;
void Init_stack(LinkStack &stack);
int Isempty_stack(LinkStack &stack);
int push_stack(LinkStack &stack,SElemtype e);
SElemtype pop_stack(LinkStack &stack);
SElemtype top_stack(LinkStack &stack);
int priority(char c);
int number_of_brackets(char* s,int length);
char* Infix_to_Postfix(char* infix,int length);
int char_match(char ch);

```

```

int calculate_Postfix(char* s,int length);
void Init_stack(LinkStack &stack){
    stack=(LinkStack)malloc(sizeof(LNode));
    stack->next=NULL;
}
int Isempty_stack(LinkStack &stack){
    return (stack->next==NULL?1:0);
}
int push_stack(LinkStack &stack,SElemtype e){
    LNode *temp=(LinkStack)malloc(sizeof(LNode));
    if(!temp){
        return -1;
    }
    temp->next=NULL;
    temp->data=e;
    temp->next=stack->next;
    stack->next=temp;
    return 1;
}
SElemtype pop_stack(LinkStack &stack){
    if(Isempty_stack(stack)){
        exit(-1);
    }else{
        LNode* temp=stack->next;
        SElemtype e;
        stack->next=temp->next;
        temp->next=NULL;
        e=temp->data;
        free(temp);
        return e;
    }
}
SElemtype top_stack(LinkStack &stack){
    if(Isempty_stack(stack)){
        exit(-1);
    }
    return stack->next->data;
}
int priority(char c){
    int res;
    switch(c){
        case '+':
            res=1;
            break;

```

```

        case '-':
            res=1;
            break;
        case '*':
            res=2;
            break;
        case '/':
            res=2;
            break;
        case '(':
            res=0;
            break;
        case '[':
            res=0;
            break;
        case '{':
            res=0;
            break;
        case '#':
            res=0;
            break;
        default:
            res=-1;
            break;
    }
    return res;
}

int number_of_brackets(char* s,int length){
    int i=0;
    int res=0;
    while(i<length){
        if(s[i]=='('||s[i]==')'||s[i]=='['||s[i]==']'||s[i]=='{'||s[i]=='}'){
            res++;
        }
        i++;
    }
    return res;
}

char* Infix_to_Postfix(char* infix,int length){
    LinkStack stack;
    Elemtyp e;
    e.type=(char*)malloc(sizeof(char));
    *(char*)e.type='#';
    Init_stack(stack);

```

```

push_stack(stack,e);
char res[1024];
char* re = (char*)malloc((length-number_of_brackets(infix,strlen(infix))) * sizeof(char));
int i=0;
int j=0;
char ch=infix[i];
while(ch!='\0'){
    switch(ch){
        case '+':
        case '-':
        case '*':
        case '/':
            while(priority(*(char*)top_stack(stack).type)>=priority(ch)){
                res[j]=*(char*)pop_stack(stack).type;
                j++;
                // i++;
                // ch=infix[i];
            }
            Elemtpe e1;
            e1.type=(char*)malloc(sizeof(char));
            *(char*)e1.type=ch;
            push_stack(stack,e1);
            i++;
            ch=infix[i];
            break;
        case '(':
        case '[':
        case '{':
            Elemtpe e2;
            e2.type=(char*)malloc(sizeof(char));
            *(char*)e2.type=ch;
            push_stack(stack,e2);
            i++;
            ch=infix[i];
            break;
        case ')':
            while(*(char*)top_stack(stack).type!= '('){
                res[j]=*(char*)pop_stack(stack).type;
                j++;
            }
            i++;
            ch=infix[i];
            pop_stack(stack);
            break;
    }
}

```

```

        case ']':
            while(*(char*)top_stack(stack).type!='['){
                res[j]=*(char*)pop_stack(stack).type;
                j++;
            }
            i++;
            ch=infix[i];
            pop_stack(stack);
            break;
        case '}':
            while(*(char*)top_stack(stack).type!='('){
                res[j]=*(char*)pop_stack(stack).type;
                j++;
            }
            pop_stack(stack);
            i++;
            ch=infix[i];
            break;
        default:
            res[j]=ch;
            i++;
            j++;
            ch=infix[i];
            break;
    }
}

```

```

    }
}

while(!Isempty_stack(stack)){
    res[j++]=*(char*)pop_stack(stack).type;
}

int index=0;
while(index<strlen(res)&&res[index]!='#'){
    re[index]=res[index];
    index++;
}

char* p=re;
return p;
}

```

```

int char_match(char ch){
    int sign=0;
    if(ch>='0'&&ch<='9'){
        sign=1;
    }
}

```

```

if(ch=='+' ){
    sign=2;
}
if(ch=='-' ){
    sign=3;
}
if(ch=='*' ){
    sign=4;
}
if(ch=='/' ){
    sign=5;
}
return sign;
}

int calculate_Postfix(char* s,int length){
LinkStack stack;
Init_stack(stack);

int index=0;
while(index<length){
    if(char_match(s[index])==1){
        Elemtpe e;
        e.type=(int*)malloc(sizeof(int));
        *(int*)e.type=(s[index]-'0');
        push_stack(stack,e);
        index++;
    }else{
        int temp2=*(int*)pop_stack(stack).type;
        int temp1=*(int*)pop_stack(stack).type;
        Elemtpe e1;
        e1.type=(int*)malloc(sizeof(int));
        if(char_match(s[index])==2){
            *(int*)e1.type=temp1+temp2;
        }
        if(char_match(s[index])==3){
            *(int*)e1.type=temp1-temp2;
        }
        if(char_match(s[index])==4){
            *(int*)e1.type=temp1*ttemp2;
        }
        if(char_match(s[index])==5){
            *(int*)e1.type=temp1/ttemp2;
        }
        push_stack(stack,e1);
    }
}
}

```

```

        index++;
    }
}

return *(int*)top_stack(stack).type;
}

int main(){
    char input[1024];
    printf("Please enter a valid arithmetic expression:\n");
    scanf("%s",input);//读取表达式
    char *pinput=input;
    printf("The expression you entered is:\n");
    printf("%s\n",pinput);
    int num=number_of_brackets(pinput,strlen(pinput));
    char *ss=new char;
    ss=Infix_to_Postfix(pinput,strlen(pinput));
    printf("The converted postfix expression is:\n");
    printf("%s\n",ss);
    printf("The expression you entered evaluates to:\n");
    printf("%d",caclulate_Postfix(ss,strlen(input)-num));
    system("pause");
    system("pause");
}

```

### 实验 2.2.2 源代码:

```

#include<iostream>
#include<string>
#include<stack>
#include<cstdlib>
#include<sstream>
#include<vector>
#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
using namespace std;

int string_to_int(string str);
int judge(string a);
vector<string> init_data(string s);
int caclulate_Postfix(vector<string> s);
vector<string> Infix_to_Postfix(vector<string> s);
void Print_Postfix(vector<string> s);
int string_to_int(string str){
    stringstream ss;
    int s;

```

```

ss<<str;
ss>>s;
return s;
}//将字符串转换为整数
int judge(string a){
    if(a=="+"||a=="-")
        return 1;
    else if(a=="*"||a=="/")
        return 2;
    else if(a=="(")
        return 3;
    else if(a==")")
        return -1;
    else
        return 0;
}//将数字的优先级定做 0, 运算符号大于 0, 便于后续处理
vector<string> init_data(string s){//初始化
    vector<string>a;
    int j,i;//定义两个快慢指针
    i=0;
    while(i<s.size())
    {
        j=i;//快指针 j 的初始位置为慢指针 i 上一次循环遍历的位置, 若循环刚开始则 i=j=0

        while(j<s.size()&&s.at(j)<='9'&&s.at(j)>='0'){
            j++;
        }//当 j 所指向的字符均是数字时, 快指针 j 增加 1
        if(i==j){
            a.push_back(s.substr(i,1));
            i=i+1;
        }//走到这里则表明没有进入上面的 while 循环, 则 i 和 j 指向的位置为非数字, 可能是符号
        else{
            a.push_back(s.substr(i,j-i));
            i=j;
        }//将多个位数的数字存储
    }
    return a;//返回整理好的字符串
}
int calculate_Postfix(vector<string> s){
    stack<int> result;
    int temp1,temp2;
    for(int i=0; i<s.size();i++){
        if(judge(s[i])==0)

```

```

        result.push(string_to_int(s[i]));

    else{
        temp1=result.top();
        result.pop();
        temp2=result.top();
        result.pop();
        if(s[i]=="+")
            result.push(temp2+temp1);

        if(s[i]=="-")
            result.push(temp2-temp1);

        if(s[i]=="*")
            result.push(temp2*temp1);

        if(s[i]=="/")
            result.push(temp2/temp1);
    }

}

return result.top();
}

vector<string> Infix_to_Postfix(vector<string> s){
    stack<string> result;
    vector<string> save;
    for(int i=0; i<s.size();i++){

        if(judge(s[i])==0)//遇到数字就直接输出
            save.push_back(s[i]);

        if(judge(s[i])>0){//如果不是数字

            if(result.empty())//栈为空则直接压栈
                result.push(s[i]);

            else{

                if(judge(s[i])>judge(result.top()))//操作符优先级比栈顶操作符优先级高
                {
                    result.push(s[i]);
                }

            }

        }

    }

}

```

```
        while(!result.empty()&&judge(result.top())>=judge(s[i])&&result.top()!="(")
    //弹栈并输出直到栈为空或遇到优先级更低的操作符（除了左括号）
    {
        save.push_back(result.top());
        result.pop();
    }
    result.push(s[i]);
}

}

if(judge(s[i])==-1){//如果遇到右括号，则全部输出括号里面的符号
    while(result.top()!="("){
        save.push_back(result.top());
        result.pop();
    }
    result.pop();
}
}

while(!result.empty()){//若此时栈仍然为空则输出所有的符号
    save.push_back(result.top());
    result.pop();
}
```

```
return save;

}

void Print_Postfix(vector<string> s){
    vector<string> res(s);
    for(int index=0;index<res.size();index++){
        if(index==res.size()-1){
            cout<<res[index]<<endl;
        }else{
            cout<<res[index]<< ",";
        }
    }
}

int main(){
    int n;
```

```
string input;//初始表达式
vector<string> str_init,str_postfix;
cout<<"please input the number of the expressions you are about to input:"<<endl;
cin>>n;
int sum=n;
while(n--){
    cout<<"%%%%%%%%%%%%%"<<endl;
    cout<<"please input the expressions:"<<endl;
    cin>>input;
    cout<<"The "<<sum-n<<" th infix-expression you entered is:"<<endl;
    cout<<input<<endl;
    str_init=init_data(input);
    str_postfix=Infix_to_Postfix(str_init);
    cout<<"The "<<sum-n<<" th postfix-expression you entered is:"<<endl;
    Print_Postfix(str_postfix);
    cout<<"The expression you entered evaluates to:"<<endl;
    cout<<caclulate_Postfix(str_postfix)<<endl;
}
system("pause");
}
```

### 三、实验环境

实验环境：

1.操作系统：

windows10

2.调试软件名称及版本号

Visual Studio Code February 2022 (version 1.65)

3.编程语言及版本号

C++/C

4.上机地点

班级未集中进行实验课

5.机器台号

华硕飞行堡垒 8

### 四、实验过程与分析

## 实验 2.1：

### 1.算法分析

主函数：

```

int main(){
    Solution_maze();
    system("pause");
}

```

主要函数：

```

void Get_maze(); // 初始化新地图
void Print_maze(); // 打印新地图
int Find_angle(int x,int y); // 寻找移动角度
void Print_stack(LinkStack &stack); // 打印栈信息
void Print_order(LinkStack &x,LinkStack &y,LinkStack &angel); // 打印路线信息
void Solution_maze(); // 处理逻辑函数

```

### (1) Get\_maze()

```

void Get_maze(){
    for(int i=0;i<col+2;i++){
        for(int j=0;j<row+2;j++){
            if(i==0||i==col+1){
                new_maze[j][i]=1;
                continue;
            }else if(j==0||j==row+1){
                new_maze[j][i]=1;
                continue;
            }else{
                new_maze[j][i]=maze[j-1][i-1];
                continue;
            }
        }
    }
}// 初始化新地图

```

此函数根据系统输入的地图初始化新地图 `new_maze`; 规定新地图的四周均为 1，中间为填入的系统输入的地图值。

时间复杂度：  $O(n^2)$ ，  $n$  为地图一维长度

空间复杂度：  $O(n^2)$

### (2) Print\_order()

```

void Print_order(LinkStack &x,LinkStack &y,LinkStack &angel){
    int index=1;
    while(!Isempty_stack(x)&&!Isempty_stack(y)){
        if(index==1){
            printf("(");
            printf("%d",*(int*)top_stack(x).type);
            printf(",");
            printf("%d",*(int*)top_stack(y).type);
            printf(",");
        }
    }
}

```

```

        printf("4");
        printf(")");
        pop_stack(x);
        pop_stack(y);
    }
    printf("<-");
    printf("(");
    printf("%d",*(int*)top_stack(x).type);
    printf(",");
    printf("%d",*(int*)top_stack(y).type);
    printf(",");
    printf("%d",*(int*)top_stack(angel).type);
    printf(")");
    pop_stack(x);
    pop_stack(y);
    pop_stack(angel);
    index++;
}
}//打印寻找出口顺序

```

此函数根据传入的栈来打印路线信息；

时间复杂度：  $O(\text{MAX}(n,m))$ ，  $n,m$  为栈  $x, y$  的长度

空间复杂度：  $O(1)$

### (3) Print\_maze()

```

void Print_maze(){
    for(int i=0;i<xlen;i++){
        for(int j=0;j<ylen;j++){
            printf("%d",new_maze[i][j]);
        }
        printf("\n");
    }
}//打印新地图

```

此函数根据新地图来打印新地图信息；

时间复杂度：  $O(n^2)$ ，  $n$  为地图一维长度

空间复杂度：  $O(1)$

### (4) Find\_angel()

```

int Find_angel(int x,int y){
    if(new_maze[x][y+1]==0&&visit[x][y+1]==0){
        return 3;
    }
    if(new_maze[x+1][y]==0&&visit[x+1][y]==0){

```

```

        return 2;
    }
    if(new_maze[x][y-1]==0&&visit[x][y-1]==0){
        return 1;
    }
    if(new_maze[x-1][y]==0&&visit[x-1][y]==0){
        return 0;
    }
    return -1;
}//寻找移动角度

```

此函数根据传入的参数(x,y)即坐标来判断下一步的行进方向；首先定义寻找路线的优先级为：右走>下走>左走>上走。如果有方向走则前进，Find\_angel(int x,int y)=3，表示向右走；Find\_angel(int x,int y)=2，表示向下走；Find\_angel(int x,int y)=1，表示向左走；Find\_angel(int x,int y)=0，表示向上走；Find\_angel(int x,int y)=-1，表示四个方向都走不通。

**时间复杂度：** O(1)

**空间复杂度：** O(1)

## (5) Solution\_maze()

```

void Solution_maze(){
    LinkStack x;
    LinkStack y;
    LinkStack angel;
    Init_stack(x);
    Init_stack(y);
    Init_stack(angel);//初始化若干栈，分别表示横坐标、纵坐标、方向坐标
    Get_maze();//初始化新地图

    int startx=1;
    int starty=1;
    int dx=startx;//横坐标
    int dy=starty;//纵坐标
    Elemtpe e1,e2;
    e1.type=(int*)malloc(sizeof(int));
    e2.type=(int*)malloc(sizeof(int));
    *(int*)e1.type=dx;
    *(int*)e2.type=dy;
    push_stack(x,e1);
    push_stack(y,e2);
    int endx=row;//定义终点的横坐标
    int endy=col;//定义终点的纵坐标
    while(*(int*)top_stack(x).type!=endx||*(int*)top_stack(y).type!=endy){//当没有达到终点时，循环寻找正确路线
        if(Find_angel(dx,dy)==3){

```

```
    dx=dx;
    dy=dy+1;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtpe eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=3;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
} //向右走
if(Find_angel(dx,dy)==2){
    dx=dx+1;
    dy=dy;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtpe eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=2;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
} //向下走
if(Find_angel(dx,dy)==1){
    dx=dx;
    dy=dy-1;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
```

```

*(int*)ex.type=dx;
*(int*)ey.type=dy;
push_stack(x,ex);
push_stack(y,ey);
Elemtpe eangel;
eangel.type=(int*)malloc(sizeof(int));
*(int*)eangel.type=1;
push_stack(angel,eangel);
if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
    break;
}
}//向左走
if(Find_angel(dx,dy)==0){
    dx=dx-1;
    dy=dy;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtpe eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=0;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
}
}//向上走
if(Find_angel(dx,dy)==-1){
    pop_stack(x);
    pop_stack(y);
    pop_stack(angel);
    dx=*(int*)top_stack(x).type;
    dy=*(int*)top_stack(y).type;
}
}//走不通了，退一步

}

printf("The maze is according to the following:\n");
Print_maze();
printf("The order is according to the following:\n");
Print_order(x,y,angel);

```

```

    printf("\n");
    printf("tips: In the output statement (x,y,angel), x represents the abscissa of the route, y
represents the ordinate of the route, and angel represents the angle of travel of the route. \n
where 0 means up, 1 means left 1, 2 means down, 3 means right, and 4 means reach the end point.");
}

}

```

此函数为本程序的主逻辑函数；首先初始化地图，给系统默认的地图四周加上墙，表示行走的路线不能越过地图边界。初始化 3 个栈，分别存储找到出口路线的横坐标 x，纵坐标 y，移动角度 angel。首先将入口(x0,y0)入栈，定义地图路线的终点(endx,endy)。开始循环，当存储横坐标栈的栈顶或者存储纵坐标的栈顶分别不等于 endx 或者 endy 时，寻找路线。定义寻找路线的优先级为：右走>下走>左走>上走。如果有方向走则前进，直到没有方向可以前进，开始后退，三个栈分别开始出栈，直到可以找到其他方向走停止出栈操作。如果最终可以到达终点，则打印出路线信息；否则程序直接退出。

**时间复杂度：**O(n)， n 为地图一维空间长度

**空间复杂度：**O(1)

## 实验 2.2:

主函数：

实验 2.2.1：

```

int main(){
    char input[1024];
    printf("Please enter a valid arithmetic expression:\n");
    scanf("%s",input);//读取表达式
    char *pinput=input;
    printf("The expression you entered is:\n");
    printf("%s\n",pinput);
    int num=number_of_brackets(pinput,strlen(pinput));
    char *ss=new char;
    ss=Infix_to_Postfix(pinput,strlen(pinput));
    printf("The converted postfix expression is:\n");
    printf("%s\n",ss);
    printf("The expression you entered evaluates to:\n");
    printf("%d",caclulate_Postfix(ss,strlen(input)-num));
    system("pause");
    system("pause");
}

```

实验 2.2.2：

```

int main(){
    int n;
    string input;
    vector<string> str_init,str_postfix;
    cout<<"please input the number of the expressions you are about to input:"<<endl;
    cin>>n;
    int sum=n;
    while(n--){
        cout<<"%%%%%%%%%%%%%"<<endl;
        cout<<"please input the expressions:"<<endl;
        cin>>input;
        cout<<"The "<<sum-n<<" th infix-expression you entered is:"<<endl;
        cout<<input<<endl;

        str_init=init_data(input);

        str_postfix=Infix_to_Postfix(str_init);
        cout<<"The "<<sum-n<<" th postfix-expression you entered is:"<<endl;
        Print_Postfix(str_postfix);

        cout<<"The expression you entered evaluates to:"<<endl;
        cout<<caclulate_Postfix(str_postfix)<<endl;
    }
    system("pause");
}

```

### 主要函数:

实验 2.2.1 中的:

```

char* Infix_to_Postfix(char* infix,int length);
int caclulate_Postfix(char* s,int length);

```

和实验 2.2.2 中的:

```

int caclulate_Postfix(vector<string> s);
vector<string> Infix_to_Postfix(vector<string> s);

```

本质上本实验最重要的步骤就是中缀表达式转换为后缀表达式和后缀表达式的计算。

#### (1) Infix\_to\_Postfix

以实验 2.2.1 的代码为例:

```

char* Infix_to_Postfix(char* infix,int length){
    LinkStack stack;
    Elemttype e;
    e.type=(char*)malloc(sizeof(char));
    *(char*)e.type='#';
    Init_stack(stack);
    push_stack(stack,e);
    char res[1024];
    char* re = (char*)malloc((length-number_of_brackets(infix,strlen(infix))) * sizeof(char));
    int i=0;
    int j=0;
    char ch=infix[i];
    while(ch!='\0'){

```

```

switch(ch){
    case '+':
    case '-':
    case '*':
    case '/':
        while(priority(*(char*)top_stack(stack).type)>=priority(ch)){
            res[j]=*(char*)pop_stack(stack).type;
            j++;
            // i++;
            // ch=infix[i];
        }
        Elemtyp e1;
        e1.type=(char*)malloc(sizeof(char));
        *(char*)e1.type=ch;
        push_stack(stack,e1);
        i++;
        ch=infix[i];
        break;
    case '(':
    case '[':
    case '{':
        Elemtyp e2;
        e2.type=(char*)malloc(sizeof(char));
        *(char*)e2.type=ch;
        push_stack(stack,e2);
        i++;
        ch=infix[i];
        break;
    case ')':
        while(*(char*)top_stack(stack).type!='('){
            res[j]=*(char*)pop_stack(stack).type;
            j++;
        }
        i++;
        ch=infix[i];
        pop_stack(stack);
        break;
    case ']':
        while(*(char*)top_stack(stack).type!='['){
            res[j]=*(char*)pop_stack(stack).type;
            j++;
        }
        i++;
        ch=infix[i];
}

```

```

        pop_stack(stack);
        break;
    case '}':
        while(*(char*)top_stack(stack).type!='('){
            res[j]=*(char*)pop_stack(stack).type;
            j++;
        }
        pop_stack(stack);
        i++;
        ch=infix[i];
        break;
    default:
        res[j]=ch;
        i++;
        j++;
        ch=infix[i];
        break;

    }
}

while(!Isempty_stack(stack)){
    res[j++]=*(char*)pop_stack(stack).type;
}

int index=0;
while(index<strlen(res)&&res[index]!='#'){
    re[index]=res[index];
    index++;
}

char* p=re;
return p;
}

```

中缀表达式  $a + b * c + (d * e + f) * g$ , 其转换成后缀表达式则为  $a\ b\ c\ * +\ d\ e\ * f\ +\ g\ * +$ 。

转换过程需要用到栈, 具体过程如下:

- 1) 如果遇到操作数, 我们就直接将其输出。
- 2) 如果遇到操作符, 则我们将其放入到栈中, 遇到左括号时我们也将其放入栈中。
- 3) 如果遇到一个右括号, 则将栈元素弹出, 将弹出的操作符输出直到遇到左括号为止。注意, 左括号只弹出并不输出。
- 4) 如果遇到任何其他的操作符, 如 “+”, “\*”, “(”) 等, 从栈中弹出元素直到遇到发现更低优先级的元素(或者栈为空)为止。弹出完这些元素后, 才将遇到的操作符压入到栈中。有一点需要注意, 只有在遇到“)”的情况下我们才弹出“(”, 其他情况我们都不会弹出“(”。
- 5) 如果我们读到了输入的末尾, 则将栈中所有元素依次弹出。

**时间复杂度:  $O(n)$ ,  $n$  为需要转换字符串的长度**

空间复杂度:  $O(n)$ , $n$  为需要转换字符串的长度

## (2) calculate\_Postfix

以实验 2.2.1 的代码为例子:

```
int calculate_Postfix(char* s,int length){
    LinkStack stack;
    Init_stack(stack);

    int index=0;
    while(index<length){
        if(char_match(s[index])==1){
            Elemtyp e;
            e.type=(int*)malloc(sizeof(int));
            *(int*)e.type=(s[index]- '0');
            push_stack(stack,e);
            index++;
        }else{
            int temp2=*(int*)pop_stack(stack).type;
            int temp1=*(int*)pop_stack(stack).type;
            Elemtyp e1;
            e1.type=(int*)malloc(sizeof(int));
            if(char_match(s[index])==2){
                *(int*)e1.type=temp1+temp2;
            }
            if(char_match(s[index])==3){
                *(int*)e1.type=temp1-temp2;
            }
            if(char_match(s[index])==4){
                *(int*)e1.type=temp1*temp2;
            }
            if(char_match(s[index])==5){
                *(int*)e1.type=temp1/temp2;
            }
            push_stack(stack,e1);
            index++;
        }
    }
    return *(int*)top_stack(stack).type;
}
```

对于后缀表达式的计算,此函数的执行过程是如果遇到数字就直接将其入栈,如果遇到符号,

则取出栈顶两个元素进行运算，然后入栈。如此循环直到到达表达式末尾，最后输出栈顶元素就是结果。

**时间复杂度：O(n),n 为需要转换字符串的长度**

**空间复杂度：O(1),函数中并没有申请额外的空间**

**算法设计巧妙之处：**对于实验 2.1，整体算法的复杂度合理。代码逻辑清晰，操作简单，容易理解。并且自定义了一个新的数据类型 SElmtype，该结构体可以实现存储任意基本数据类型的功能，使得代码具有一定的普适性，提高了代码的复用性。依据题意要求，栈作为本实验的基本存储结构。对于路线信息的存储分别采用三个维度(横坐标，纵坐标，方向)，是的存储结构清晰。符合题意要求的同时保留了实验的创新性。对于实验 2.2，在编写完实验 2.2.1 的代码时，发现不能完成多位数的运算。但是其事先定义字符优先级、计算后缀表达式计算的同时对存储栈进行修改的实验思路值得借鉴，因此完成了实验 2.2.2 的代码(为了实验代码的简洁，因为在实验 2.2.1 中已经使用了自定义数据结构栈的应用，因而此处使用了 STL)。实验 2.2.2 在实验 2.2.1 上作出了可以进行多位数后缀表达式的转换和输入多个表达式进行求值的若干改进。整体算法的复杂度合理。代码逻辑清晰，操作简单，容易理解。

**可扩展性：**本实验的抽象数据类型的实现具有一定的可扩展性。栈可以实现任意基本类型的存储。

## 2. 调试过程

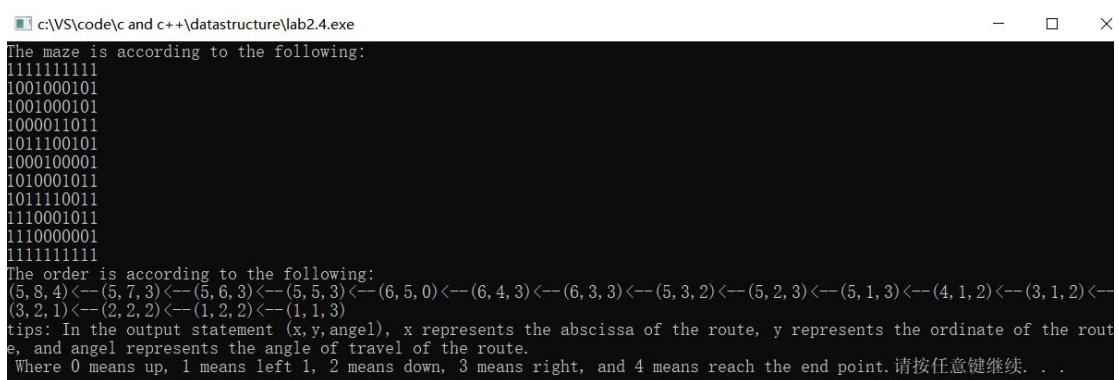
### 实验 2.1：

#### 1.

问题代码：

```
int endx=row;//定义终点的横坐标
int endy=col;//定义终点的纵坐标
while(*(int*)top_stack(x).type!=endx&&*(int*)top_stack(y).type!=endy){//当没有达到终点时，循环寻找正确路线}
```

错误运行结果：



```
c:\VS\code\c and c++\datastructure\lab2.4.exe
The maze is according to the following:
1111111111
1001000101
1001000101
1000011011
1011100101
1000100001
1010001011
1011110011
1110001011
1110000001
1111111111
The order is according to the following:
(5, 8, 4)---(5, 7, 3)---(5, 6, 3)---(5, 5, 3)---(6, 5, 0)---(6, 4, 3)---(6, 3, 3)---(5, 3, 2)---(5, 2, 3)---(5, 1, 3)---(4, 1, 2)---(3, 1, 2)---(3, 2, 1)---(2, 2, 2)---(1, 2, 2)---(1, 1, 3)
tips: In the output statement (x, y, angel), x represents the abscissa of the route, y represents the ordinate of the route, and angel represents the angle of travel of the route.
Where 0 means up, 1 means left, 2 means down, 3 means right, and 4 means reach the end point. Please press any key to continue. . .
```

发现结果是没有运行到出口就程序终止了。原因是 while 循环的条件出错。

代码修改为：

```
int endx=row;//定义终点的横坐标
int endy=col;//定义终点的纵坐标
while(*(int*)top_stack(x).type!=endx||*(int*)top_stack(y).type!=endy){//当没有达到终点时，循环寻找正确路线}
```

正确运行结果：

```
c:\VS\code\c and c++\datastructure\lab2.4.exe
The maze is according to the following:
111111111
1001000101
1001000101
1000011011
101100101
1000100001
1010001011
1011110011
110001011
1100000001
1111111111
The order is according to the following:
(9, 8, 4) <--(9, 7, 3) <--(8, 7, 2) <--(7, 7, 2) <--(6, 7, 2) <--(5, 7, 2) <--(5, 6, 3) <--(5, 5, 3) <--(6, 5, 0) <--(6, 4, 3) <--(6, 3, 3) <--(5, 3, 2) <--(5, 2, 3) <--(5, 1, 3) <--(4, 1, 2) <--(3, 1, 2) <--(3, 2, 1) <--(2, 2, 2) <--(1, 2, 2) <--(1, 1, 3)
tips: In the output statement (x, y, angel), x represents the abscissa of the route, y represents the ordinate of the route, and angel represents the angle of travel of the route.
Where 0 means up, 1 means left, 2 means down, 3 means right, and 4 means reach the end point. Please press any key to continue. . .
```

2.

问题代码:

```
if(Find_angel(dx,dy)==3){
    dx=dx;
    dy=dy+1;
    visit[dx][dy]=1;
    Elemtypex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtypeeangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=3;
    push_stack(angel,eangel);
```

运行结果:

程序直接闪退，程序被强制关闭。

发现是以下代码的原因:

```

SElemtype pop_stack(LinkStack &stack){
    if(Iempty_stack(stack)){
        exit(-1);
    }else{
        LNode* temp=stack->next;
        SElemtype e;
        stack->next=temp->next;
        temp->next=NULL;
        e=temp->data;
        free(temp);
        return e;
    }
}//出栈

SElemtype top_stack(LinkStack &stack){
    if(Iempty_stack(stack)){
        // Elemtpe e;
        // e.type=(int*)malloc(sizeof(int));
        // *(int*)e.type=-1;
        // return e;
        exit(-1);
    }
    return stack->next->data;
}
}//取出栈顶元素

```

是因为栈在运行结束后全部为空，这两个对于栈的操作如果栈为空的话会直接 exit。经过 debug 发现，在找到出口后，程序仍然会继续执行 while 中的 4 个 if 语句，没有及时判断是否不满足 while 循环的条件。应该将代码修改为：

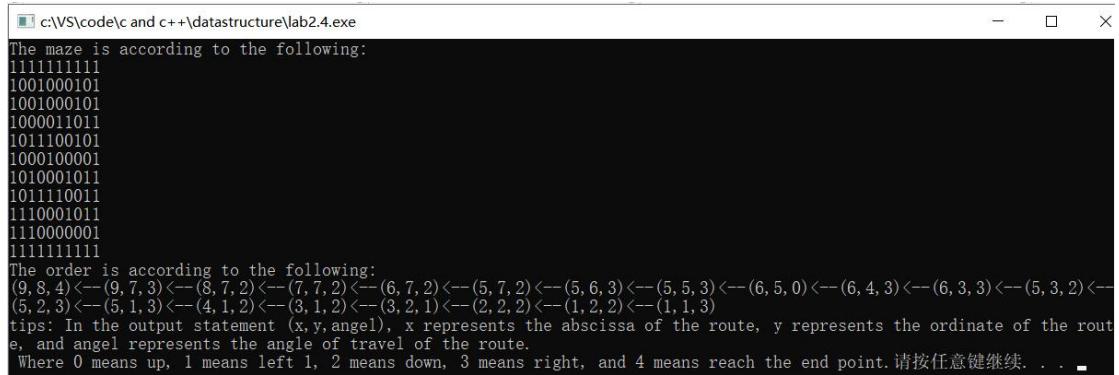
```

if(Find_angel(dx,dy)==3){
    dx=dx;
    dy=dy+1;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtpe eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=3;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
}
}//向右走

```

在每个 if 语句中，如果对坐标进行了修改，则判断是否可以跳出循环。

正确运行结果：



```
c:\VS\code\c and c++\datastructure\lab2.4.exe
The maze is according to the following:
1111111111
1001000101
1001000101
1000011011
1011100101
1000100001
1010001011
1011110011
1100001011
1110000001
1111111111
The order is according to the following:
(9, 8, 4) <--> (9, 7, 3) <--> (8, 7, 2) <--> (7, 7, 2) <--> (6, 7, 2) <--> (5, 6, 3) <--> (5, 5, 3) <--> (6, 5, 0) <--> (6, 4, 3) <--> (6, 3, 3) <--> (5, 3, 2) <--> (5, 2, 3) <--> (5, 1, 3) <--> (4, 1, 2) <--> (3, 1, 2) <--> (3, 2, 1) <--> (2, 2, 2) <--> (1, 2, 2) <--> (1, 1, 3)
tips: In the output statement (x, y, angel), x represents the abscissa of the route, y represents the ordinate of the route, and angel represents the angle of travel of the route.
Where 0 means up, 1 means left 1, 2 means down, 3 means right, and 4 means reach the end point. 请按任意键继续. . .
```

## 实验 2.2:

1.

问题代码：

```
int main()
{
    char input[1024];
    printf("Please enter a valid arithmetic expression:\n");
    scanf("%s",input); //读取表达式
    char *pinput=input;
    printf("The expression you entered is:\n");
    printf("%s\n",pinput);
    int num=number_of_brackets(pinput,strlen(pinput));
    char *ss=new char;
    ss=Infix_to_Postfix(pinput,strlen(pinput));
    printf("The converted postfix expression is:\n");
    printf("%s\n",ss);
    printf("The expression you entered evaluates to:\n");
    printf("%d",caclulate_Postfix(ss,strlen(input))); //需要减去括号的数量！！！ -num
    system("pause");
    system("pause");
}
```

在 main 函数中计算后缀表达式的传入参数中，如果传入的表达式包含括号，会导致传入的实际后缀表达式长度不符合。程序出现直接闪退的情况，通过 debug 发现是如下代码导致的闪退：

```

SElemtype pop_stack(LinkStack &stack){
    if(Iempty_stack(stack)){
        exit(-1);
    }else{
        LNode* temp=stack->next;
        SElemtype e;
        stack->next=temp->next;
        temp->next=NULL;
        e=temp->data;
        free(temp);
        return e;
    }
}

SElemtype top_stack(LinkStack &stack){
    if(Iempty_stack(stack)){
        exit(-1);
    }
    return stack->next->data;
}

```

因此猜测原因可能是栈为空，而函数 int calculate\_Postfix(char\* s,int length)的返回值为栈顶的元素，因此栈为空导致 exit 的发生，判断是循环次数出错。

这里贴上函数 int calculate\_Postfix(char\* s,int length)的代码：

```

int calculate_Postfix(char* s,int length){
    LinkStack stack;
    Init_stack(stack);
    int index=0;
    while(index<length){
        if(char_match(s[index])==1){
            Elemtpe e;
            e.type=(int*)malloc(sizeof(int));
            *(int*)e.type=(s[index]-'0');
            push_stack(stack,e);
            index++;
        }else{
            int temp2=*(int*)pop_stack(stack).type;
            int temp1=*(int*)pop_stack(stack).type;
            Elemtpe e1;
            e1.type=(int*)malloc(sizeof(int));
            if(char_match(s[index])==2){
                *(int*)e1.type=temp1+temp2;
            }
            if(char_match(s[index])==3){
                *(int*)e1.type=temp1-temp2;
            }
        }
    }
}

```

```

    if(char_match(s[index])==4){
        *(int*)e1.type=temp1*temp2;
    }
    if(char_match(s[index])==5){
        *(int*)e1.type=temp1/temp2;
    }
    push_stack(stack,e1);
    index++;
}

}
return *(int*)top_stack(stack).type;
}

```

通过前面中缀表达式转换为后缀表达式，发现后缀表达式是没有“(”，“)”，“{”，“}”，“[”，“]”，因此中缀表达式转换为后缀表达式的过程可能会导致长度变小，因此编写了

```
int number_of_brackets(char* s,int length);
```

来计算中缀表达式括号的数量。

对应一行代码修改为：

```
printf("%d",caclulate_Postfix(ss,strlen(input)-num));
```

运行结果为：

```
c:\VS\code\c and c++\datastructure\lab2.3.exe
Please enter a valid arithmetic expression:
(6+2*(3+6*(6+6)))
The expression you entered is:
(6+2*(3+6*(6+6)))
The converted postfix expression is:
623666+**+*?
The expression you entered evaluates to:
156请按任意键继续. . .
```

实验结果正确，达到实验预期。

## 五、实验结果总结

回答以下问题：

(1) 你的测试充分吗？为什么？你是怎样考虑的？

答：对于实验 2.1，测试比较充分；

实验结果：

```
c:\VS\code\c and c++\datastructure\lab2.4.exe
The maze is according to the following:
111111111
1001000101
1001000101
1000011011
1011001011
1000100001
1010001011
1011110011
1100001011
1100000001
1111111111
The order is according to the following:
(9, 8, 4) <--(9, 7, 3) <--(8, 7, 2) <--(7, 7, 2) <--(6, 7, 2) <--(5, 7, 2) <--(5, 6, 3) <--(5, 5, 3) <--(6, 5, 0) <--(6, 4, 3) <--(6, 3, 3) <--(5, 3, 2) <--(5, 2, 3) <--(5, 1, 3) <--(4, 1, 2) <--(3, 1, 2) <--(3, 2, 1) <--(2, 2, 2) <--(1, 2, 2) <--(1, 1, 3)
tips: In the output statement (x, y, angel), x represents the abscissa of the route, y represents the ordinate of the route, and angel represents the angle of travel of the route.
Where 0 means up, 1 means left, 2 means down, 3 means right, and 4 means reach the end point. Please press any key to continue. . .
```

路线信息合理且唯一。

对于**实验 2.2**，测试充分，以下为实验结果：

首先输入需要输入的表达式的数量：

```
please input the number of the expressions you are about to input:
6
```

然后是若干个表达式：

```
%%%%%%%%%%%%%
please input the expressions:
3*(7-2)
The 1 th infix-expression you entered is:
3*(7-2)
The 1 th postfix-expression you entered is:
3, 7, 2, -, *
The expression you entered evaluates to:
15
%%%%%%%%%%%%%
please input the expressions:
8
The 2 th infix-expression you entered is:
8
The 2 th postfix-expression you entered is:
8
The expression you entered evaluates to:
8
%%%%%%%%%%%%%
please input the expressions:
1+2+3+4
The 3 th infix-expression you entered is:
1+2+3+4
The 3 th postfix-expression you entered is:
1, 2, +, 3, +, 4, +
The expression you entered evaluates to:
10
%%%%%%%%%%%%%
```

```
%%%%%
please input the expressions:
88-1*5
The 4 th infix-expression you entered is:
88-1*5
The 4 th postfix-expression you entered is:
88, 1, 5, *, -
The expression you entered evaluates to:
83
%%%%%
```

```
%%%%%
please input the expressions:
1024/4*8
The 5 th infix-expression you entered is:
1024/4*8
The 5 th postfix-expression you entered is:
1024, 4, /, 8, *
The expression you entered evaluates to:
2048
%%%%%
```

```
%%%%%
please input the expressions:
(6+2*(3+6*(6+6)))
The 6 th infix-expression you entered is:
(6+2*(3+6*(6+6)))
The 6 th postfix-expression you entered is:
6, 2, 3, 6, 6, 6, +, *, +, *, +
The expression you entered evaluates to:
156
请按任意键继续. . . ■
```

均满足测试样例。对于本实验的测试样例，无非就是从以下几个特殊情况考虑：输入单个字符，输入若干括号组合的表达式，输入多位整数的表达式，以上的组合.....发现实验结果均达到预期。

(2) 为什么你要选用栈或队列或字符串或数组等抽象数据类型作为你应用的数据结构？

答：栈具有先进后出的特点。在**实验 2.1** 的实验过程中，选用栈十分便于处理数据。因为涉及到路线信息的存储，而这些信息存储到数据结构里很可能是暂时的，栈的结构特点十分适合处理这些信息。**在实验 2.2 中**，将中缀表达式转换为后缀表达式的过程需要用到栈，因为当从左到右扫描时不能确定当前运算符的优先级别（没有扫到最右边，还不能预知）。

(3) 用一段简短的代码及说明论述你的应用中主要的函数的主要处理部分。

答：对于**实验 2.1**，在函数 `Solution_maze()` 中：

```
while(*(int*)top_stack(x).type!=endx||*(int*)top_stack(y).type!=endy){//当没有达到终点时，循环寻找正确路线
    if(Find_angle(dx,dy)==3){
        dx=dx;
        dy=dy+1;
    }
}
```

```
visit[dx][dy]=1;
Elemtpe ex,ey;
ex.type=(int*)malloc(sizeof(int));
ey.type=(int*)malloc(sizeof(int));
*(int*)ex.type=dx;
*(int*)ey.type=dy;
push_stack(x,ex);
push_stack(y,ey);
Elemtpe eangel;
eangel.type=(int*)malloc(sizeof(int));
*(int*)eangel.type=3;
push_stack(angel,eangel);
if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
    break;
}
}//向右走
if(Find_angel(dx,dy)==2){
    dx=dx+1;
    dy=dy;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
    push_stack(x,ex);
    push_stack(y,ey);
    Elemtpe eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=2;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
}
}//向下走
if(Find_angel(dx,dy)==1){
    dx=dx;
    dy=dy-1;
    visit[dx][dy]=1;
    Elemtpe ex,ey;
    ex.type=(int*)malloc(sizeof(int));
    ey.type=(int*)malloc(sizeof(int));
    *(int*)ex.type=dx;
    *(int*)ey.type=dy;
```

```

    push_stack(x,ex);
    push_stack(y,ey);
    Elemtpe eangel;
    eangel.type=(int*)malloc(sizeof(int));
    *(int*)eangel.type=1;
    push_stack(angel,eangel);
    if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
        break;
    }
    }//向左走
    if(Find_angel(dx,dy)==0){
        dx=dx-1;
        dy=dy;
        visit[dx][dy]=1;
        Elemtpe ex,ey;
        ex.type=(int*)malloc(sizeof(int));
        ey.type=(int*)malloc(sizeof(int));
        *(int*)ex.type=dx;
        *(int*)ey.type=dy;
        push_stack(x,ex);
        push_stack(y,ey);
        Elemtpe eangel;
        eangel.type=(int*)malloc(sizeof(int));
        *(int*)eangel.type=0;
        push_stack(angel,eangel);
        if(*(int*)top_stack(x).type==endx&&*(int*)top_stack(y).type==endy){
            break;
        }
    }
    }//向上走
    if(Find_angel(dx,dy)==-1){
        pop_stack(x);
        pop_stack(y);
        pop_stack(angel);
        dx=*(int*)top_stack(x).type;
        dy=*(int*)top_stack(y).type;
    }
    }//走不通了，退一步
}

```

如果不满足 while 循环的条件（栈顶元素不是终点），判断能否向右走，如果能进入其 if 语句，对横坐标+1，纵坐标不变，然后判断是否到达终点，如果到达则退出循环，否则继续判断能否进行下一个 if 语句；判断能否向下走，如果能进入其 if 语句，对纵坐标+1，横坐标不变，然后判断是否到达终点，如果到达则退出循环，否则继续判断能否进行下一个 if 语句；判断能否向左走，如果能进入其 if 语句，对横坐标-1，纵坐标不变，然后判断是否到达终点，如果到达则退出循环，否则继续判断能否进行

下一个 if 语句；判断能否向上走，如果能进入其 if 语句，对纵坐标-1，横坐标不变，然后判断是否到达终点，如果到达则退出循环，否则继续判断能否进行下一个 if 语句；判断是否四个方向都走不了，如果能进入其 if 语句，开始退栈操作。如此反复循环，直到到达终点。

## 实验 2.2 中，函数

```
char* Infix_to_Postfix(char* infix,int length)
```

中，核心代码如下：

```
char ch=infix[i];  
while(ch!='\0'){  
    switch(ch){  
        case '+':  
        case '-':  
        case '*':  
        case '/':  
            while(priority(*(char*)top_stack(stack).type)>=priority(ch)){  
                res[j]=*(char*)pop_stack(stack).type;  
                j++;  
            }  
            Elemtyp e1;  
            e1.type=(char*)malloc(sizeof(char));  
            *(char*)e1.type=ch;  
            push_stack(stack,e1);  
            i++;  
            ch=infix[i];  
            break;  
        case '(':  
        case '[':  
        case '{':  
            Elemtyp e2;  
            e2.type=(char*)malloc(sizeof(char));  
            *(char*)e2.type=ch;  
            push_stack(stack,e2);  
            i++;  
            ch=infix[i];  
            break;  
        case ')':  
            while(*(char*)top_stack(stack).type!=')'{  
                res[j]=*(char*)pop_stack(stack).type;  
                j++;  
            }  
            i++;  
            ch=infix[i];  
            pop_stack(stack);  
            break;  
    }  
}
```

```

        case ']':
            while(*(char*)top_stack(stack).type!='['){
                res[j]=*(char*)pop_stack(stack).type;
                j++;
            }
            i++;
            ch=infix[i];
            pop_stack(stack);
            break;
        case '}':
            while(*(char*)top_stack(stack).type!='('){
                res[j]=*(char*)pop_stack(stack).type;
                j++;
            }
            pop_stack(stack);
            i++;
            ch=infix[i];
            break;
        default:
            res[j]=ch;
            i++;
            j++;
            ch=infix[i];
            break;
    }

}

while(!IsEmpty_stack(stack)){
    res[j++]=*(char*)pop_stack(stack).type;
}

```

首先取出表达式的第一个字符 ch，并提前申明一个辅助栈 stack 和存储最终表达式的字符数组 res，开始判断字符进入 res 的顺序。提前将字符“#”入栈，其优先级为 0。如果 ch 是运算符 (+, -, \*, /)，循环判断栈顶元素和 ch 的优先级，如果前者大于后者则一直将栈顶部元素赋值给 res 并出栈；如果是左括号，则直接入栈；如果是右括号，则栈顶弹出顶部元素赋值给 res 直到遇到匹配的左括号。如果是数字，也直接入栈。最后如果字符串循环结束，若栈中还存在元素，则将栈顶元素弹出赋值给 res，直到栈为空。

(4) 你的应用中采用的是顺序的还是链式的存储结构？为什么要选用这种存储结构？

答：顺序栈：采用顺序存储结构可以模拟栈存储数据的特点，从而实现栈存储结构；链式栈：采用链式存储结构实现栈结构。两种实现方式的区别，仅限于数据元素在实际物理空间上存放的相对位置，顺序栈底层采用的是数组，链栈底层采用的是链表。本实验选用的是链式栈。顺序存储结构：优点：连续存储，空间利用率高；缺点：不

方便数据的增删。链式存储结构：优点：对于数据的增删比较方便；缺点：浪费空间。在本实验中，不需要考虑空间的开销。而且链式栈便于操作，易于实现，具有普适性。因而在**实验 2.1 和 2.2** 中，均选择链式栈。

(5) 源程序的大致的执行过程是怎样的？

答：对于**实验 2.1**，首先初始化地图，给系统默认的地图四周加上墙，表示行走的路线不能越过地图边界。初始化 3 个栈，分别存储找到出口路线的横坐标 x，纵坐标 y，移动角度 angel。首先将入口(x0,y0)入栈，定义地图路线的终点(endx,endy)。开始循环，当存储横坐标栈的栈顶或者存储纵坐标的栈顶分别不等于 endx 或者 endy 时，寻找路线。定义寻找路线的优先级为：右走>下走>左走>上走。如果有方向走则前进，直到没有方向可以前进，开始后退，直到可以找到其他方向走。如果最终可以到达终点，则打印出路线信息；否则程序直接退出。

对于**实验 2.2**，首先定义字符的优先级，然后将输入的中缀表达式转换为后缀表达式，最后将后缀表达式进行计算并输出结果。其中最重要的是将输入的中缀表达式转换为后缀表达式。中缀表达式  $a + b*c + (d * e + f) * g$ ，其转换成后缀表达式则为  $a\ b\ c\ * +\ d\ e\ * f\ +\ g\ * +$ 。转换过程需要用到栈，具体过程如下：如果遇到操作数，我们就直接将其输出；如果遇到操作符，则我们将其放入到栈中，遇到左括号时我们也将其放入栈中；如果遇到一个右括号，则将栈元素弹出，将弹出的操作符输出直到遇到左括号为止。注意，左括号只弹出并不输出；如果遇到任何其他的操作符，如（“+”，“\*”，“（”）等，从栈中弹出元素直到遇到发现更低优先级的元素（或者栈为空）为止。弹出完这些元素后，才将遇到的操作符压入到栈中。有一点需要注意，只有在遇到“）”的情况下我们才弹出“（”，其他情况下我们都不会弹出“（”；如果我们读到了输入的末尾，则将栈中所有元素依次弹出。然后是后缀表达式的计算，运行过程是如果遇到数字就直接将其入栈，如果遇到符号，则取出栈顶两个元素进行运算，然后入栈。如此循环直到到达表达式末尾，最后输出栈顶元素就是结果。

## 六、附录

(1) 如果你对这个实验还有其他的解决方案或设想，或对我们的实验方案有什么意见，请在此描述：

对于**实验 2.1**，可以适当增加一些数据样例。

(2) 实验参考的资料：

**实验 2.1：**暂无

**实验 2.2：**参考数据结构 PPT 栈与队列部分的前缀，中缀和后缀表达式内容。

(3) 回答思考题

a) 栈和队列在计算机系统中有哪些应用？写出你知道的系统中，这两种抽象数据类型的应用。

答：在对高级语言编写的源程序进行编译时，类似于表达式括号匹配问题就是使用栈来解决的；计算机系统在处理子程序之间的调用关系是，用栈来保存处理执行过程中的调用次序。程序设计中，也经常使用队列记录需按照先进先出方式处理的数据，例如键盘缓冲区，操作系统中的作业调度等。例子是主机和打印机速度不匹配，这个时候会有一个缓冲区，主机把一堆要打印的东西放到缓冲区，然后打印机去缓冲区拿东西打印。这样主机就不用等打印机慢吞吞，放完东西就可以走。第二个例子是 CPU 里边的资源竞争，多个终端或者进程都请求占用 CPU，但是 CPU 只有一个，怎么做？排队，此外为了

防止队伍前边的进程占用过多时间导致后边的进程排到资源遥遥无期，每个进程会有一个执行时间限制，执行不完也会换下一个。

- b) 在程序调用的时候，需要进行函数的切换，你认为函数在进行切换时系统要做那些工作？

答：首先将需要切换出来的函数和返回参数出栈，然后将不用的函数和返回参数压栈。

(4) 选作：查询以下内容的有关知识

- a) 函数调用、返回时，系统对栈进行的操作。

答：1. 指令指针的地址+1，指向函数调用后的下一条指令。这个地址会被记入堆栈中，它将作为函数返回时的返回地址。

2. 在堆栈中给声明的返回值建立空间。

3. 当前堆栈的栈顶被记录下来并存储在一个叫做栈帧的特殊指针中，从记录到函数运行结束加入堆栈的所有数据都视为函数的局部变量。

4. 函数将所有形参都放入到堆栈中。

5. 开始执行指令指针中的指针。

6. 局部变量被压入堆栈中。

7. 当函数返回时，返回值放入步骤 2 所建立的内存中，随后堆栈指针指向栈帧指针，从而依次弹出被调函数的所有局部变量并将返回值弹出堆栈，赋值给函数调用本身的值，返回函数返回地址，继续运行下面程序。

b) 堆的管理都要做那些工作？具体怎样做的？

答：windows 堆管理是建立在虚拟内存管理的基础之上的，每个进程都有独立的 4GB 的虚拟地址空间，其中有 2GB 的属于用户区，保存的是用户程序的数据和代码，而系统在装载程序时会将这部分内存划分为 4 个段从低地址到高地址依次为静态存储区，代码段，堆段和栈段，其中堆的生长方向是从低地址到高地址，而栈的生长方向是从高地址到低地址。

程序申请堆内存时，系统会在虚拟内存的基础上分配一段内存，然后记录下来这块的大小和首地址，并且在对应内存块的首尾位置各有相应的数据结构，所以在堆内存上如果发生缓冲区溢出的话，会造成程序崩溃，这部分没有硬件支持，所有管理算法都有开发者自己设计实现。

堆内存管理的函数主要有 HeapCreate、HeapAlloc、HeapFree、  
HeapRealloc、HeapDestroy、HeapWalk、HeapLock、HeapUnLock。

堆内存的分配与释放

堆内存的分配主要用到函数 HeapAlloc，下面是这个函数的原型：

```
LPVOID HeapAlloc(  
    HANDLE hHeap, //堆句柄，表示在哪个堆上分配内存  
    DWORD dwFlags, //分配的内存的相关标志  
    DWORD dwBytes //大小  
) ;
```

堆句柄可以使用进程默认堆也可以使用用户自定义的堆，自定义堆使用函数 HeapCreate，函数返回堆的句柄，使用 GetProcessHeap 可以获取系统默认堆，返回的也是一个堆句柄。分配内存的相关标志有这样几个值：

**HEAP\_NO\_SERIALIZE:** 这个表示对堆内存不进行线程并发控制，由于系统默认会进行堆的并发控制，防止多个线程同时分配到了同一个堆内存，如果程序是单线程程序则可以添加这个选项，适当提高程序运行效率。

**HEAP\_ZERO\_MEMORY:** 这个标志表示在分配内存的时候同时将这块内存清零。

HeapCreate 函数的原型如下：

```
HANDLE HeapCreate(
    DWORD f1Options, //堆的相关属性
    DWORD dwInitialSize, //堆初始大小
    DWORD dwMaximumSize //堆所占内存的最大值
);
```

f1Options 的取值如下：

**HEAP\_NO\_SERIALIZE:** 取消并发控制

**HEAP\_SHARED\_READONLY:** 其他进程可以以只读属性访问这个堆

**dwInitialSize, dwMaximumSize** 这两个值如果都是 0，那么堆内存的初始大小由系统分配，并且堆没有上限，会根据具体的需求而增长。下面是使用的例子：

```
//在系统默认堆中分配内存
srand((unsigned int)time(NULL));
HANDLE hHeap = GetProcessHeap();
int nCount = 1000;
float *pfArray = (float *)HeapAlloc(hHeap, HEAP_ZERO_MEMORY |
    HEAP_NO_SERIALIZE, nCount * sizeof(float));
for (int i = 0; i < nCount; i++)
{
    pfArray[i] = 1.0f * rand();
}

HeapFree(hHeap, HEAP_NO_SERIALIZE, pfArray);

//在自定义堆中分配内存
hHeap = HeapCreate(HEAP_GENERATE_EXCEPTIONS, 0, 0);
pfArray = (float *)HeapAlloc(hHeap, HEAP_ZERO_MEMORY |
    HEAP_NO_SERIALIZE, nCount * sizeof(float));
for (int i = 0; i < nCount; i++)
{
    pfArray[i] = 1.0f * rand();
}

HeapFree(hHeap, HEAP_NO_SERIALIZE, pfArray);
HeapDestroy(hHeap);
```

2. 遍历进程中所有堆的信息：

便利堆的信息主要用到函数 HeapWalk，该函数的原型如下：

```
BOOL WINAPI HeapWalk(
```

```
    __in          HANDLE hHeap, //堆的句柄
    __in_out      LPPPROCESS_HEAP_ENTRY lpEntry //返回堆内存的相关
信息
);
```

下面是 PROCESS\_HEAP\_ENTRY 的原型：

```
typedef struct _PROCESS_HEAP_ENTRY {
    PVOID lpData;
    DWORD cbData;
    BYTE cbOverhead;
    BYTE iRegionIndex;
    WORD wFlags;
    union {
        struct {
            HANDLE hMem;
            DWORD dwReserved[3];
            } Block;
        struct {
            DWORD dwCommittedSize;
            DWORD dwUnCommittedSize;
            LPVOID lpFirstBlock;
            LPVOID lpLastBlock;
            } Region;
        };
} PROCESS_HEAP_ENTRY, *LPPPROCESS_HEAP_ENTRY;
```

下面是遍历堆内存的例子：

```
PHANDLE pHeaps = NULL;
//当传入的参数为 0 和 NULL 时，函数返回进程中堆的个数
int nCount = GetProcessHeaps(0, NULL);
pHeaps = new HANDLE[nCount];
//获取进程所有堆句柄
GetProcessHeaps(nCount, pHeaps);
PROCESS_HEAP_ENTRY phe = {0};
for (int i = 0; i < nCount; i++)
{
    cout << "Heap handle: 0x" << pHeaps[i] << '\n';
    //在读取堆中的相关信息时需要将堆内存锁定，防止程序向堆
```

中写入数据

```
    HeapLock(pHeaps[i]);
    HeapWalk(pHeaps[i], &phe);
    //输出堆信息
    cout << "\tSize: " << phe.cbData << " - Overhead: "
        << static_cast<DWORD>(phe.cbOverhead) << '\n';
    cout << "\tBlock is a";
    if (phe.wFlags & PROCESS_HEAP_REGION)
```

```

{
    cout << " VMem region:\n";
    cout << "\tCommitted size: " <<
phe.Region.dwCommittedSize << '\n';
    cout << "\tUncommitted size: " <<
phe.Region.dwUnCommittedSize << '\n';
    cout << "\tFirst block: 0x" <<
phe.Region.lpFirstBlock << '\n';
    cout << "\tLast block: 0x" << phe.Region.lpLastBlock
<< '\n';
}
else
{
    if(phe.wFlags & PROCESS_HEAP_UNCOMMITTED_RANGE)
    {
        cout << "n uncommitted range\n";
    }
    else if(phe.wFlags & PROCESS_HEAP_ENTRY_BUSY)
    {
        cout << "n Allocated range: Region index - "
            << static_cast<unsigned>(phe.iRegionIndex)
<< '\n';
        if(phe.wFlags & PROCESS_HEAP_ENTRY_MOVEABLE)
        {
            cout << "\tMovable: Handle is 0x" <<
phe.Block.hMem << '\n';
        }
        else if(phe.wFlags &
PROCESS_HEAP_ENTRY_DDESHARE)
        {
            cout << "\tDDE Sharable\n";
        }
    }
    else cout << " block, no other flags specified\n";
}
cout << std::endl;
HeapUnlock(pHeaps[i]);
ZeroMemory(&phe, sizeof(PROCESS_HEAP_ENTRY));
}
delete[] pHeaps;

```

---

# 实验报告

课程名称：数据结构与算法分析 班级：软信 2001 实验成绩：  
实验名称：分治策略 学号：20207130 批阅教师签字：  
实验编号：实验三 姓名：赖骏鸿 实验日期：2020 年 5 月 26 日  
指导教师：马毅 组号：暂无 实验时间：18 时 30 分— 22 时 20 分

## 一、实验目的

- 1.理解分治法的思想。
- 2.掌握使用分治法解决问题。

## 二、实验内容

本实验选择了 Gray 码问题(记作实验 3.1),归并排序问题(记作实验 3.2)。

### 1.Gray 码问题 (记作实验 3.1)

#### ★ 问题描述

Gray 码是一个长度为  $2^n$  的序列。序列中无相同的元素，每个元素都是长度为 n 位的串，相邻元素恰好只有一位不同。用分治策略设计一个算法对任意的 n 构造相应的 Gray 码。

#### ★ 编程任务

利用分治策略试设计一个算法对任意的 n 构造相应的 Gray 码。

#### ★ 数据输入

由文件 input.txt 提供输入数据 n。

#### ★ 结果输出

程序运行结束时，将得到的所有编码输出到文件 output.txt 中。

#### 输入文件示例

input.txt

3

#### 输出文件示例

output.txt

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

#### ★ 实现提示

把原问题分解为两个子问题，分别对两个子问题的每个数组后一位加 0 和 1。

### 2.归并排序(记作实验 3.2)

#### ★ 问题描述

目前的网上拍卖系统会显示很多待拍卖的物品，通常这些系统具有按照某个关

关键字对打出的广告进行排序列出的功能，并且能够按照用户输入的某个关键字进行过滤，找到某些特定的物品。

### ★ 编程任务

定义一个 `Advertisement` 类，该类中至少包含该物品的数量，名称，联系人 e-mail，最好有开拍时间及关闭时间，根据用户输入的关键字比如名称， mail，时间等，利用非递归的归并排序对所有的广告进行排序，并列出所有排好序的广告。

### ★ 数据输入

由文件 `input.txt` 提供输入的所有广告信息。程序中由用户输入要排序的关键字。

### ★ 结果输出

程序运行结束时，排好序的广告输出到文件 `output.txt` 中，并为每个广告添加序号。

输入文件示例	输出文件示例
<code>input.txt</code>	<code>output.txt</code>
Coat (物品名称)	1
3 (数量)	Bag
<a href="mailto:a@mail.com">a@mail.com</a>	12
Skirt	<a href="mailto:a@mail.com">a@mail.com</a>
5	2
<a href="mailto:b@mail.com">b@mail.com</a>	Cap
Cap	7
7	<a href="mailto:c@mail.com">c@mail.com</a>
<a href="mailto:c@mail.com">c@mail.com</a>	3
Bag	Coat (物品名称)
12	3 (数量)
<a href="mailto:a@mail.com">a@mail.com</a>	<a href="mailto:a@mail.com">a@mail.com</a>
Title (用户输入按照 title 排序)	4
	Skirt
	5
	<a href="mailto:b@mail.com">b@mail.com</a>

## 3. 中位数问题(记作实验 3. 3)

### ★ 问题描述

设  $X[0 : n - 1]$  和  $Y[0 : n - 1]$  为两个数组，每个数组中含有  $n$  个已排好序的数。找出  $X$  和  $Y$  的  $2n$  个数的中位数。

### ★ 编程任务

利用分治策略试设计一个  $O(\log n)$  时间的算法求出这  $2n$  个数的中位数。

### ★ 数据输入

由文件 `input.txt` 提供输入数据。文件的第 1 行中有 1 个正整数  $n$  ( $n \leq 200$ )，表示每个数组有  $n$  个数。接下来的两行分别是  $X$ ,  $Y$  数组的元素。

### ★ 结果输出

程序运行结束时，将计算出的中位数输出到文件 output.txt 中。

#### 输入文件示例

input.txt

3

5 15 18

3 14 21

#### 输出文件示例

output.txt

14

### ★ 实现提示

比较两个序列的中位数大小，如果两个数相等，则该数为整个  $2n$  个数据的中位数，否则通过比较，分别减少两个序列的查找范围，确定查找的起止位置，继续查找。

### 三、实验环境

实验环境：

操作系统：

windows10

调试软件名称及版本号：

Visual Studio Code February 2022 (version 1.65)

编程语言及版本号：

C++/C

上机地点：

班级未集中进行实验课

机器台号：

华硕飞行堡垒 8

### 四、问题分析

(1) 分析需要解决的问题，并给出你的思路，可以借助图表等辅助表达。

答：在实验 3.1 中，题目要求的是采用分治法解决 Gray 码问题。运用分治递归求解  $n$  位的 Gray 码的思路为：将求解  $n$  位 Gray 码的问题划分成求解  $n-1$  位 Gray 码的问题，再将求解  $n-1$  位 Gray 码的问题划分成求解  $n-2$  位 Gray 码的问题……直到划分成求解 1 位 Gray 码的问题，1 位 Gray 码为 0 和 1，通过 1 位 Gray 码构造出 2 位格雷码，再通过 2 位 Gray 码构造出 3 位 Gray 码……直到构造出  $n$  位 Gray 码。参考反射法，从  $n-1$  位 Gray 码构造  $n$  位 Gray 码的思路为：把  $2^n$  个 Gray 码分成两部分；把前  $2^{n-1}$  个 Gray 码的最高位置 0，其余位与  $n-1$  位 Gray 码一一对应，无需更改；把后  $2^{n-1}$  个 Gray 码的最高位置 1，其余位由  $n-1$  位 Gray 码上下翻转而来。图表如下图：

记  $n$  为 Gray 码位数

求解  $n=1 \Rightarrow n=2$ :

$\begin{matrix} 0 \\ 1 \end{matrix} \Rightarrow \begin{matrix} 0 & 0 \\ 0 & 1 \\ 1 & \end{matrix}$

最高位的前面一半补 0  
后面一半补 1

$\begin{matrix} 0 & 0 \\ 0 & 1 \\ 1 & \end{matrix} \Rightarrow \begin{matrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{matrix}$

其余位的前面一半  
由前面一半翻转而来

求解:  $n=2 \Rightarrow n=3$

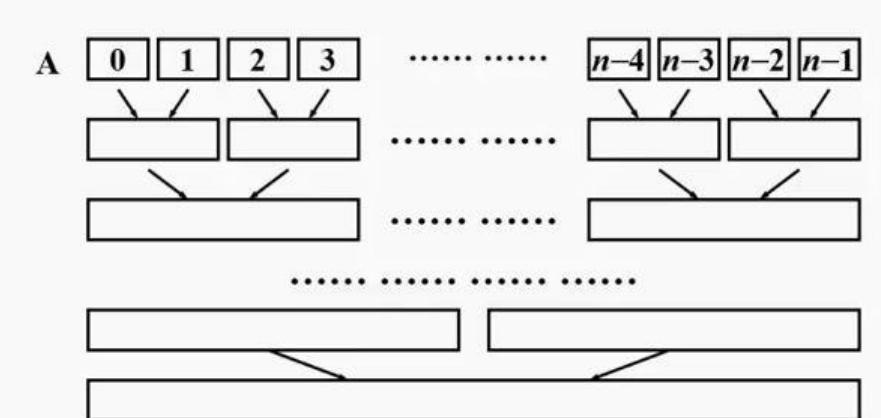
$\begin{matrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{matrix} \Rightarrow \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & \end{matrix}$

最高位的前面一半补 0  
后面一半补 1

$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & \end{matrix} \Rightarrow \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{matrix}$

其余位的前面一半  
由前面一半翻转而来

在实验 3.2 中，题目的要求时采用非递归的归并排序方法对 advertisement 类进行排序，依照的标准是 advertisement 的若干属性。思路如下：首先封装好 advertisement 类，然后根据用户输入的要求不同，利用分治法的思想，对若干 advertisement 进行非递归的归并排序。非递归的归并排序思路如下：非递归就是先从最底层开始先把数组分割成每个子数组包含两个元素，利用 merge 方法，对这个子数组进行排序，等这些子数组都排序完成，然后下一轮就是将数组分割成每个子数组包含 4 个元素，再排序 ··· 按照 1、2、4、8 ··· 顺序来分割，直到最后只剩下一个数组排序完成。那么对于所有的排序均完成。具体过程如下图所示：



在实验 3.3 中，题目并未要求有递归或者非递归方法的限制。因此经过对问题的分析我们采用递归法。首先计算两个数组的中位数，再根据中位数的大小来确定目标中位数的查找范围，最终找到目标中位数。

对于已经排序好的数组：若  $n$  为奇数，则数组下标为  $(n-1)/2$  处的数即为中位数，若  $n$  为偶数，则取  $(n-1)/2$  向下取整和向上取整这两个位置的数的平均值作为中位数。然后，比较两个序列的中位数大小：(1) 若  $m_1 = m_2$ ，返回结果，该数为整个  $2n$  个数据的中位数；(2) 若  $m_1 < m_2$ ，则表明中位数在  $(m_1, m_2)$  之间，即整个  $2n$  的数的中位数在 X 数组的后一半和 Y 数组的前一半中；(3) 若  $m_1 > m_2$ ，则表明则中位数在  $(m_2, m_1)$

之间，即整个  $2n$  的数的中位数在 X 数组的 前一半和 Y 数组的后一半中；若出现  $m_1 = m_2$  的情况，算法结束，否则通过比较，分别减少两个序列的查找范围，确定查找的起止位置，继续查找，直至数组分割至左右两部数组只有一个数字的情况，求其平均值即为中位数。

(2) 分析利用你的想法可能会有怎么样的时空复杂度？

答：在实验 3.1 中，对于时间复杂度：

递归函数 void Cmry\_code (int bits) 的时间复杂度为：

$$T(n) = \begin{cases} O(1) & n=1 \\ T(n-1) + O(2^n \cdot n) \end{cases}$$

因此

$$\begin{aligned} T(n) &= T(n-1) + O(2^n \cdot n) = T(n-1) + n \cdot 2^n O(1) \\ &= O(1) + (2 \cdot 2^2 + \dots + n \cdot 2^n) \cdot O(1) \\ &= O(1) [1 + (2n-1) \cdot 2^n] \\ &= O(n \cdot 2^n) \end{aligned}$$

对于空间复杂度：因为最终数组的有效位数为  $n \cdot 2^n$ ，因而空间复杂度为  $O(n \cdot 2^n)$

时间复杂度： $O(n \cdot 2^n)$

空间复杂度： $O(n \cdot 2^n)$

在实验 3.2 中，对于时间复杂度：

非递归并排序算法的实现是这样的：对于一个特定的输入 length (length 从 1 开始递增)，函数 Merge\_sort\_non\_recursive 调用函数 Merge\_pass  $\log n$  次，函数 Merge\_pass 调用函数 Merge  $n / (2 * \text{length})$  次，而 Merge 函数自身调用 length 次。因此归并排序的时间复杂度为  $O(n \log n)$ 。

对于空间复杂度：非递归版的归并排序，省略了中间的栈空间，直接申请一段  $O(n)$  的地址空间即可，因此空间复杂度为  $O(n)$ 。

时间复杂度： $O(n \log n)$

空间复杂度： $O(n)$

在实验 3.3 中，该算法每次将子问题减半（至多差常数 1）。时间复杂度满足方程：

$O(n) = O(n/2) + 1$ ，并且  $O(1) = 1$ ；

最终解得  $O(n) = \log n$ 。由于需要读取的数据规模为  $n$ ，因此空间复杂度为  $O(n)$ 。

时间复杂度： $O(\log n)$

空间复杂度： $O(n)$

(3) 描述你再进行实现时，主要的函数或操作内部的主要算法；分析这个算法的时空复杂度，并说明你设计的巧妙之处，如有创新，将其清晰的表达。

答：在实验 3.1 中，主函数：

```
int main(){
    string inputfilename="L://LAB/input_lab3.2.txt";
    string outputfilename="L://LAB/output_lab3.2.txt";
    int bits=Open_file(inputfilename);
    Gray_code(bits);
    Write_file(outputfilename,bits);
    Print_Gray_code(bits);
    system("pause");
}
```

主要函数：

函数	说明
void Write_file(string filename,int bits)	写入文件函数
int Open_file(string filename)	打开文件并读取里面的整数值
void Gray_code(int bits)	Gray 码问题解决逻辑函数

1) Write\_file

代码部分：

```
void Write_file(string filename,int bits){
    ofstream of(filename);
    int number=(int)Get_power(bits);
    for(int i=0;i<number;i++){
        for(int j=0;j<bits;j++){
            of<<gray[i][bits-j-1]<<" ";
        }
        of<<endl;
    }
    of.close();
}
```

此函数主要满足对有效位格雷码的输出，并写在文件中。

时间复杂度：  $O(n \cdot 2^n)$

空间复杂度：  $O(1)$

2) Open\_file

代码部分：

```

int Open_file(string filename){
    ifstream in_file;
    int bits=0;
    in_file.open(filename);
    string item;
    getline(in_file,item);
    while(in_file){
        getline(in_file,item);
        bits=stoi(item);
    }
    return bits;
}

```

此函数主要完成从文件中读取整数。

时间复杂度:  $O(1)$

空间复杂度:  $O(1)$

### 3) Gray\_code

代码部分:

```

void Gray_code(int bits){
    int number=(int)Get_power(bits);
    if(bits==1){
        gray[0][0]=0;
        gray[1][0]=1;
        return;
    }
    for(int i=0;i<number/2;i++){
        gray[i][bits-1]=0;
        gray[number-1-i][bits-1]=1;
    }

    Gray_code(bits-1);
    for(int i=number/2;i<number;i++){
        for(int j=0;j<bits-1;j++){
            gray[i][j]=gray[number-i-1][j];
        }
    }
}

```

此函数完成对有效位 Gray 码的数据填充。

上问题已经分析过此函数的时空复杂度:

时间复杂度:  $O(n \cdot 2^n)$

空间复杂度:  $O(n \cdot 2^n)$

在实验 3.2 中:

主函数:

```

int main(){
    Utils m;
    vector<Advertisement> ads;
    m.open_file_solution("L://LAB/input.txt",ads);
    int choice;
    cout<<endl;
    cout<<"***** Welcome to the advertising system *****"<<endl;
    cout<<"***** 0 for exit" &gt;&gt;choice;
    cout<<"***** 1 for amount" &gt;&gt;choice;
    cout<<"***** 2 for name" &gt;&gt;choice;
    cout<<"***** 3 for email" &gt;&gt;choice;
    cout<<"***** 4 for start_time" &gt;&gt;choice;
    cout<<"***** 5 for end_time" &gt;&gt;choice;
    cout<<"*****" &gt;&gt;choice;
    cout<<"please input your choice: ";
    cin>>choice;
    if(choice!=0){
        m.Merge_sort_recursive(ads,0,ads.size()-1,choice);
        m.Write_file_solution("L://LAB/output.txt",ads);
        cout<<"please observe the results in the txt file: output.txt"<<endl;
    }else{
        cout<<"Exit the system successfully!"<<endl;
    }
    system("pause");
}

```

### 主要函数：

函数	说明
<b>void Merge(vector&lt;T&gt; &amp;A,int p,int q,int r,int choice)</b>	归并排序算法
<b>bool Compare(vector&lt;T&gt; &amp;L,vector&lt;T&gt; &amp;R,int l,int r,int choice)</b>	比较范式数据结构中元素的大小
<b>void Merge_pass(vector&lt;T&gt; &amp;A,int N, int length,int choice)</b>	分段归并排序
<b>void Merge_sort_non_recursive(vector&lt;T&gt; &amp;A, int N,int choice)</b>	非递归归并排序的实现

1) void Merge

```

template<class T>
void Merge(vector<T> &A,int p,int q,int r,int choice){
    int len_L=q-p+1;
    int len_R=r-q;
    int max=INT_MAX;
    vector<T> L;
    vector<T> R;
    vector<T> res;
    for(int i=0;i<len_L;i++){
        L.push_back(A[p+i]);
    }
    for(int i=0;i<len_R;i++){
        R.push_back(A[q+1+i]);
    }
    Advertisement ad(INT_MAX,INT_MAX,"~","~","~","~");
    L.push_back(ad);
    R.push_back(ad);
    int indexi=0;
    int indexj=0;
    for(int k=p;k<r+1;k++){
        if(Compare(L,R,indexi,indexj,choice)){
            A[k]=L[indexi];
            indexi++;
        }else{
            A[k]=R[indexj];
            indexj++;
        }
    }
}

```

此函数完成对指定子数组的归并排序。此函数的设计思路是通过系统传入的参数  $p, q, r$  将数组  $A$  分割为两个子数组  $A[p \dots q]$  和  $A[q+1 \dots r]$ , 将两个子数组分别存储到  $\text{vector}$  数组  $L$  和  $R$  中。

然后本算法设计的巧妙之处来了：在  $L$  和  $R$  数组中的末尾插入一个数据，此数据为无穷大，由于本实验是需要对类  $\text{advertisement}$  进行排序，因而存入的  $\text{advertisement}$  类属性均为无穷大。这样在下面通过逐个比较数组  $L$  和  $R$  中元素的大小，取出其中小的元素到原数组  $A$  中进行存储。由于  $L$  和  $R$  中已经事先插入了一个无穷大的数据，所以  $L$  和  $R$  在循环过程中永远不可能为空。简化了代码复杂度。

时间复杂度:  $O(n)$ ,  $n$  为数组  $A$  的长度

空间复杂度:  $O(n)$ ,  $n$  为数组  $A$  的长度

2) bool Compare

```

template<class T>
bool Compare(vector<T> &L,vector<T> &R,int l,int r,int choice){
    bool sign=false;
    switch(choice){
        case 1:
            if(L[l].getAmount()<=R[r].getAmount()){
                sign=true;
            }
            break;
        case 2:
            if(L[l].getName()<=R[r].getName()){
                sign=true;
            }
            break;
        case 3:
            if(L[l].getE_mail()<=R[r].getE_mail()){
                sign=true;
            }
            break;
        case 4:
            if(L[l].getStart_time()<=R[r].getStart_time()){
                sign=true;
            }
            break;
        case 5:
            if(L[l].getEnd_time()<=R[r].getEnd_time()){
                sign=true;
            }
            break;
    }
    return sign;
}

```

此函数主要实现对于不同的 choice 值，对 advertisement 类根据不同属性进行比较元素的大小。

时间复杂度： $O(1)$

空间复杂度： $O(1)$

3) void Merge\_pass

```

template<class T>
void Merge_pass(vector<T> &A, int N, int length, int choice)
{ /* 两两归并相邻有序子列 */
    int i, j;
    for ( i=0; i <= N-2*length; i += 2*length ){
        Merge( A, i, i+length-1, i+2*length-1, choice );
    }
    if ( i+length < N ){
        Merge( A, i, i+length-1, N-1, choice );
    } /* 归并最后2个子列 */
}

```

此函数主要实现对于数组根据特定区间长度 length 的分段归并排序。

时间复杂度:  $O(n)$ ,  $n$  为数组的长度

空间复杂度:  $O(1)$

#### 4) void Merge\_sort\_non\_recursive

```

template<class T>
void Merge_sort_non_recursive(vector<T> &A, int N, int choice)
{
    int length;
    length = 1;
    while( length < N ) {
        Merge_pass( A, N, length, choice );
        length *= 2;
    }
}

```

此函数是非递归归并排序的主逻辑函数。

时间复杂度:  $O(\log n)$ ,  $n$  为数组的长度

空间复杂度:  $O(1)$

在实验 3.3 中:

主函数:

```

int main() {
    int n;
    string infilename, outfilename;
    infilename = "L://LAB/input_lab3.1.txt";
    outfilename = "L://LAB/output_lab3.1.txt";
    Open_File(infilename, n, X, Y);
    double res = Count_Median(X, Y, n);
    write_File(outfilename, res);
    cout << "The result is: " << res << endl;
    system("pause");
}

```

主要函数:

函数	说明
----	----

---

```
double Get_Median(int* A, int n)
```

获取数组 A 的中位数

```
double Count_Median(int* X, int* Y, int n)
```

获取数组 X 和 Y 的中位数

---

1) double Get\_Median

```
double Get_Median(int* A, int n) {  
    double m = 0;  
    if (n % 2 == 0) { //若数组中数据个数为偶数  
        m = (A[n / 2 - 1] + A[n / 2]) / 2.0;  
    }  
    else { //若数组中数据个数为奇数  
        m = A[n / 2];  
    }  
    return m; //返回浮点数类型  
}
```

此函数主要完成获取数组 A 的中位数的功能。

2) double Count\_Median

```
double Count_Median(int* X, int* Y, int n) { //计算两个有序数组的中位数  
    double x = Get_Median(X, n);  
    double y = Get_Median(Y, n);  
    double z = 0; //z 为最终两个有序数组的中位数  
    if (n == 1) { //直接计算两个数的平均值为中位数  
        z = (x + y) / 2;  
        return z;  
    }else if (n == 2) {  
        z = (max(X[0], Y[0]) + min(X[1], Y[1])) / 2;  
        return z;  
    }  
    if (x == y) {  
        z = x;  
        return z;  
    }else if (x < y) { //舍去 X 中小于 x 的所有前半部分数据以及 Y 中大于 y 的所有后半部分数据, //得到新的数组 X1,Y1 和数组大小 n1  
        if (n % 2 == 0) { //当 n 为偶数时  
            //保留 X 中最后一个小于 x 的数和 Y 中第一个大于 y 的数, 即保留原来计算中位数的两个数  
            int* X1 = X + n / 2 - 1; //数组 X1 取 X 的下标为 n/2-1 的数及其之后所有数  
            int* Y1 = Y; //数组 Y1 取 Y 中下标为 0 直到下标为 n/2 的数  
            int n1 = n - n / 2 + 1;  
            return Count_Median(X1, Y1, n1);  
        }else {  
            int* X1 = X + (n - 1) / 2; //数组 X1 取 X 的下标为(n-1)/2 的数及其之后所有数  
            int* Y1 = Y; //数组 Y1 取 Y 中下标为 0 直到下标为(n-1)/2-1 的数  
            int n1 = n - (n - 1) / 2;  
            return Count_Median(X1, Y1, n1);  
        }  
    }
```

```

    }

    }else if (x > y) { //操作思路和上述相同

        if (n % 2 == 0) {

            int* X2 = X;
            int* Y2 = Y + n / 2 - 1;
            int n2 = n - n / 2 + 1;

            return Count_Median(X2, Y2, n2);

        }else {

            int* X2 = X;
            int* Y2 = Y + (n - 1) / 2;
            int n2 = n - (n - 1) / 2;

            return Count_Median(X2, Y2, n2);
        }
    }

    return -1;
}

```

此函数主要完成对于两个数组求取中位数的操作。

对于已经排序好的数组：若  $n$  为奇数，则数组下标为  $(n-1)/2$  处的数即为中位数，若  $n$  为偶数，则取  $(n-1)/2$  向下取整和向上取整这两个位置的数的平均值作为中位数。然后，比较两个序列的中位数大小：(1) 若  $m1 = m2$ , 返回结果, 该数为整个  $2n$  个数据的中位数；(2) 若  $m1 < m2$ , 则表明中位数在  $(m1, m2)$  之间, 即整个  $2n$  的数的中位数在  $X$  数组的后一半和  $Y$  数组的前一半中；(3) 若  $m1 > m2$ , 则表明则中位数在  $(m2, m1)$  之间, 即整个  $2n$  的数的中位数在  $X$  数组的前一半和  $Y$  数组的后一半中；若出现  $m1 = m2$  的情况, 算法结束, 否则通过比较, 分别减少两个序列的查找范围, 确定查找的起止位置, 继续查找, 直至数组分割至左右两部数组只有一个数字的情况, 求其平均值即为中位数。

(4) 你在调试过程中发现了怎样的问题? 又作出了怎样的改进?

答：对于**实验 3.1**，在调试过程中进展较为顺利，暂时未发现任何问题。

对于**实验 3.2**，在调试过程中主要是对归并排序算法的测试出现一些问题，主要是上述函数入口的不精确造成。暂时未发现一些较大问题。对于**实验 3.3**，一开始未考虑到中位数可能是浮点数的情况，其他暂时未发现较大问题。

## 五、实验结果总结

回答以下问题：

(1) 对不同的输入，该算法存在哪几类可能出现的情况，你的测试充分吗？为什么？你是怎样考虑的？

答：对于**实验 3.1**，我认为我的测试十分充分。

测试一：

输入：

input\_lab3.2.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

4

输出：

```
c:\VS\code\c and c++  
0 0 0 0  
0 0 0 1  
0 0 1 1  
0 0 1 0  
0 1 1 0  
0 1 1 1  
0 1 0 1  
0 1 0 0  
1 1 0 0  
1 1 0 1  
1 1 1 1  
1 1 1 0  
1 0 1 0  
1 0 1 1  
1 0 0 1  
1 0 0 0  
请按任意键继续. . .
```

```
output_lab3.2.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
0 0 0 0  
0 0 0 1  
0 0 1 1  
0 0 1 0  
0 1 1 0  
0 1 1 1  
0 1 0 1  
0 1 0 0  
1 1 0 0  
1 1 0 1  
1 1 1 1  
1 1 1 0  
1 0 1 0  
1 0 1 1  
1 0 0 1  
1 0 0 0
```

测试二:

输入:

```
input_lab3.2.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
1
```

输出:

```
c:\VS\code\c and c++\d  
0  
1  
请按任意键继续. . .
```

---

```
output_lab3.2.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
0  
1
```

实验均达到预期。

对于**实验 3.2:**

输入：

```
input.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
Coat  
3  
a@mail.com  
09:00  
22:00  
Skirt  
5  
b@mail.com  
09:01  
22:01  
Cap  
7  
c@mail.com  
09:02  
22:02  
Bag  
12  
a@mail.com  
09:03  
22:03
```

测试一：

```
*****  
please input the way you want to sort the advertisements:  
The Advertisement has the following attributes:  
amount, name, email, start_time, end_time  
  
%%%%%%%%%%%%% Welcome to the advertising system %%%%%%%%%%%%%%  
***** 0 for exit *****  
***** 1 for amount *****  
***** 2 for name *****  
***** 3 for email *****  
***** 4 for start_time *****  
***** 5 for end_time *****  
%%%%%%%%%%%%%  
please input your choice: 1  
please observe the results in the txt file: output.txt  
请按任意键继续. . .
```

按照广告数量排序。

输出：

output.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
Ad Order:1  
Ad id:1  
Ad amount:3  
Ad name:Coat  
Ad email:a@mail.com  
Ad start\_time:09:00  
Ad end\_time:22:00  
Ad Order:2  
Ad id:2  
Ad amount:5  
Ad name:Skirt  
Ad email:b@mail.com  
Ad start\_time:09:01  
Ad end\_time:22:01  
Ad Order:3  
Ad id:3  
Ad amount:7  
Ad name:Cap  
Ad email:c@mail.com  
Ad start\_time:09:02  
Ad end\_time:22:02  
Ad Order:4  
Ad id:4  
Ad amount:12  
Ad name:Bag  
Ad email:a@mail.com  
Ad start\_time:09:03  
Ad end\_time:22:03

测试二:

```
*****  
please input the way you want to sort the advertisements:  
The Advertisement has the following attributes:  
amount, name, email, start_time, end_time  
  
%%%%%%%%%%%%% Welcome to the advertising system %%%%%%%  
***** 0 for exit *****  
***** 1 for amount *****  
***** 2 for name *****  
***** 3 for email *****  
***** 4 for start_time *****  
***** 5 for end_time *****  
%%%%%%  
please input your choice: 2  
please observe the results in the txt file: output.txt  
请按任意键继续. . .
```

按照广告名字排序。

输出：

output.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Ad Order:1  
Ad id:4  
Ad amount:12  
Ad name:Bag  
Ad email:a@mail.com  
Ad start\_time:09:03  
Ad end\_time:22:03  
Ad Order:2  
Ad id:3  
Ad amount:7  
Ad name:Cap  
Ad email:c@mail.com  
Ad start\_time:09:02  
Ad end\_time:22:02  
Ad Order:3  
Ad id:1  
Ad amount:3  
Ad name:Coat  
Ad email:a@mail.com  
Ad start\_time:09:00  
Ad end\_time:22:00  
Ad Order:4  
Ad id:2  
Ad amount:5  
Ad name:Skirt  
Ad email:b@mail.com  
Ad start\_time:09:01  
Ad end\_time:22:01

测试三:

```
*****  
please input the way you want to sort the advertisements:  
The Advertisement has the following attributes:  
amount, name, email, start_time, end_time
```

```
%%%%%%%%%%%%%% Welcome to the advertising system %%%%%%%%%%%%%%  
***** 0 for exit *****  
***** 1 for amount *****  
***** 2 for name *****  
***** 3 for email *****  
***** 4 for start_time *****  
***** 5 for end_time *****  
%%%%%%%%%%%%%
```

```
please input your choice: 3
```

```
please observe the results in the txt file: output.txt
```

```
请按任意键继续...
```

```
按照广告电子邮箱排序。
```

输出：

output.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
Ad Order:1  
Ad id:1  
Ad amount:3  
Ad name:Coat  
Ad email:a@mail.com  
Ad start\_time:09:00  
Ad end\_time:22:00  
Ad Order:2  
Ad id:4  
Ad amount:12  
Ad name:Bag  
Ad email:a@mail.com  
Ad start\_time:09:03  
Ad end\_time:22:03  
Ad Order:3  
Ad id:2  
Ad amount:5  
Ad name:Skirt  
Ad email:b@mail.com  
Ad start\_time:09:01  
Ad end\_time:22:01  
Ad Order:4  
Ad id:3  
Ad amount:7  
Ad name:Cap  
Ad email:c@mail.com  
Ad start\_time:09:02  
Ad end\_time:22:02

测试四：

```
*****  
please input the way you want to sort the advertisements:  
The Advertisement has the following attributes:  
amount, name, email, start_time, end_time  
  
%%%%%%%%%%%%% Welcome to the advertising system %%%%%%%  
***** 0 for exit *****  
***** 1 for amount *****  
***** 2 for name *****  
***** 3 for email *****  
***** 4 for start_time *****  
***** 5 for end_time *****  
%%%%%%  
please input your choice: 4  
please observe the results in the txt file: output.txt  
请按任意键继续. . .
```

按照广告的开始时间排序。

输出：

output.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
Ad Order:1  
Ad id:1  
Ad amount:3  
Ad name:Coat  
Ad email:a@mail.com  
Ad start\_time:09:00  
Ad end\_time:22:00  
Ad Order:2  
Ad id:2  
Ad amount:5  
Ad name:Skirt  
Ad email:b@mail.com  
Ad start\_time:09:01  
Ad end\_time:22:01  
Ad Order:3  
Ad id:3  
Ad amount:7  
Ad name:Cap  
Ad email:c@mail.com  
Ad start\_time:09:02  
Ad end\_time:22:02  
Ad Order:4  
Ad id:4  
Ad amount:12  
Ad name:Bag  
Ad email:a@mail.com  
Ad start\_time:09:03  
Ad end\_time:22:03

测试五：

output.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Ad Order:1
Ad id:1
Ad amount:3
Ad name:Coat
Ad email:a@mail.com
Ad start_time:09:00
Ad end_time:22:00
Ad Order:2
Ad id:2
Ad amount:5
Ad name:Skirt
Ad email:b@mail.com
Ad start_time:09:01
Ad end_time:22:01
Ad Order:3
Ad id:3
Ad amount:7
Ad name:Cap
Ad email:c@mail.com
Ad start_time:09:02
Ad end_time:22:02
Ad Order:4
Ad id:4
Ad amount:12
Ad name:Bag
Ad email:a@mail.com
Ad start_time:09:03
Ad end_time:22:03
```

实验均达到预期！

对于实验 3.3，我认为测试也十分充分，测试如下：

测试一：

输入：

input\_lab3.1.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
3
5 15 18
3 14 21
```

程序运行结果：

```
c:\VS\code\c and c++\datastructure\lab3_1.exe
The result is: 14
请按任意键继续. . .
```

输出：

```
output_lab3.1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
14
```

测试二：

输入：

```
input_lab3.1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
5
3 6 8 10 13
4 6 13 17 29
```

运行结果：

```
c:\VS\code\c and c++\datastructure\lab3_1.exe
The result is: 9
请按任意键继续. . .
```

输出：

```
output_lab3.1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
9
```

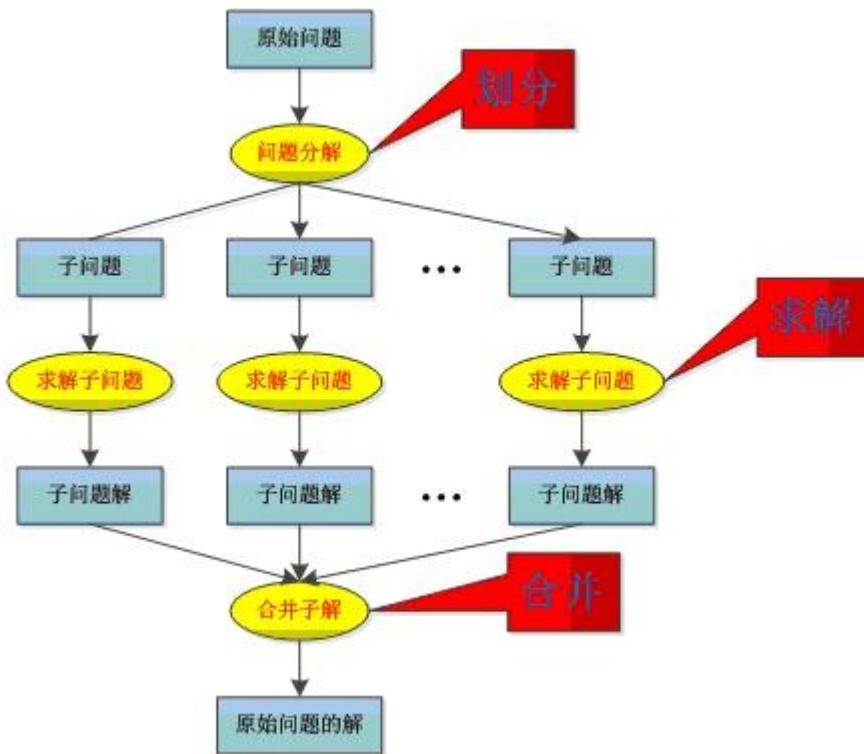
实验均达到预期！

(2) 所选用的数据结构合适吗？

答：在**实验 3.1** 中，采用数组，合适。在**实验 3.2** 中，采用带有模板函数的 vector 数组，合适。在**实验 3.2** 中，采用数组，合适。

(3) 叙述通过实验你对分治法的理解及你认为的分治法的优缺点。

答：分治，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。如图所示：



分治算法的优缺点：

优点：可以使得代码简洁化，思路逻辑清晰。结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性。

缺点：如果子问题不独立，需要重复求公共子问题，算法效率低下。

## 六、附录

(1) 如果你对这个实验还有其他的解决方案或设想，或对我们的实验方案有什么意见，请在此描述。

**实验 3.1:**

暂无

**实验 3.2:**

暂无

**实验 3.3:**

暂无

(2) 实验参考的资料

**实验 3.1:**

[https://blog.csdn.net/qq\\_50737715/article/details/123587206?spm=1001.2014.3001.5506](https://blog.csdn.net/qq_50737715/article/details/123587206?spm=1001.2014.3001.5506)

**实验 3.2:**

[https://blog.csdn.net/jason\\_cuijiahui/article/details/79038468?ops\\_request\\_misc=%257B%2522request%255Fid%2522%253A%2522165389828116780366573788%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\\_id=165389828116780366573788&biz\\_id=0&utm\\_medium=distribute.pc\\_search\\_result.none-task-blog-2~all~baidu\\_landing\\_](https://blog.csdn.net/jason_cuijiahui/article/details/79038468?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522165389828116780366573788%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=165389828116780366573788&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~baidu_landing_)

v2~default-1-79038468-null-null.142^v11^control,157^v12^control&utm\_term=c%2B%2Bstring%E6%AF%94%E8%BE%83%E5%A4%A7%E5%B0%8F&utm\_id=1018.2226.3001.4187

[https://blog.csdn.net/weixin\\_42524410/article/details/105015591](https://blog.csdn.net/weixin_42524410/article/details/105015591)

### 实验 3. 3:

暂无

(3) 实验源代码:

### 实验 3. 1:

```
#include<iostream>
#include<string>
#include<stack>
#include<cstdlib>
#include<sstream>
#include<vector>
#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
#include <limits.h>
#include<fstream>
#include<cmath>
using namespace std;

int gray[1024][1024];
int Get_power(int bits){
    return pow(2,bits);
}
void Gray_code(int bits){
    int number=(int)Get_power(bits);
    if(bits==1){
        gray[0][0]=0;
        gray[1][0]=1;
        return;
    }
    for(int i=0;i<number/2;i++){
        gray[i][bits-1]=0;
        gray[number-1-i][bits-1]=1;
    }
    Gray_code(bits-1);
    for(int i=number/2;i<number;i++){
        for(int j=0;j<bits-1;j++){
            gray[i][j]=gray[number-i-1][j];
        }
    }
}
```

```

}

void Print_Gray_code(int bits){
    int number=(int)Get_power(bits);
    for(int i=0;i<number;i++){
        for(int j=0;j<bits;j++){
            cout<<gray[i][bits-j-1]<<"   ";
        }
        cout<<endl;
    }
}

int Open_file(string filename){
    ifstream in_file;
    int bits=0;
    in_file.open(filename);
    string item;
    getline(in_file,item);
    while(in_file){
        getline(in_file,item);
        bits=stoi(item);
    }
    return bits;
}

void Write_file(string filename,int bits){
    ofstream of(filename);
    int number=(int)Get_power(bits);
    for(int i=0;i<number;i++){
        for(int j=0;j<bits;j++){
            of<<gray[i][bits-j-1]<<"   ";
        }
        of<<endl;
    }
    of.close();
}

int main(){
    string inputfilename="L://LAB/input_lab3.2.txt";
    string outputfilename="L://LAB/output_lab3.2.txt";
    int bits=Open_file(inputfilename);
    Gray_code(bits);
    Write_file(outputfilename,bits);
    Print_Gray_code(bits);
    system("pause");
}

```

## 实验 3.2:

```
#include<iostream>
#include<string>
#include<stack>
#include<cstdlib>
#include<sstream>
#include<vector>
#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
#include <limits.h>
#include<fstream>
using namespace std;

class Advertisement{
private:
    int id;
    int amount;
    string name;
    string e_mail;
    string start_time;
    string end_time;
public:
    Advertisement(int id,int amount,string name,string e_mail,string start_time,string end_time){
        this->id=id;
        this->amount=amount;
        this->name=name;
        this->e_mail=e_mail;
        this->start_time=start_time;
        this->end_time=end_time;
    }
    // ~Advertisement(){
    //     cout<<"class Advertisement is destroyed....."<<endl;
    // }
    int getId(){
        return this->id;
    }
    int getAmount(){
        return this->amount;
    }
    string getName(){
        return this->name;
    }
}
```

```

    }

    string getE_mail(){
        return this->e_mail;
    }

    string getStart_time(){
        return this->start_time;
    }

    string getEnd_time(){
        return this->end_time;
    }

};

class Utils{
public:
    template<class T>
    void Merge(vector<T> &A,int p,int q,int r,int choice){
        int len_L=q-p+1;
        int len_R=r-q;
        int max=INT_MAX;
        vector<T> L;
        vector<T> R;
        vector<T> res;
        for(int i=0;i<len_L;i++){
            L.push_back(A[p+i]);
        }
        for(int i=0;i<len_R;i++){
            R.push_back(A[q+1+i]);
        }
        Advertisement ad(INT_MAX,INT_MAX,"~","~","~","~");
        L.push_back(ad);
        R.push_back(ad);
        int indexi=0;
        int indexj=0;
        for(int k=p;k<r+1;k++){
            if(Compare(L,R,indexi,indexj,choice)){
                A[k]=L[indexi];
                indexi++;
            }else{
                A[k]=R[indexj];
                indexj++;
            }
        }
    }

    // "0 for exit,1 for amount,2 for name,3 for email,4 for start_time,5 for end_time"
    template<class T>

```

```

bool Compare(vector<T> &L,vector<T> &R,int l,int r,int choice){
    bool sign=false;
    switch(choice){
        case 1:
            if(L[l].getAmount()<=R[r].getAmount()){
                sign=true;
            }
            break;
        case 2:
            if(L[l].getName()<=R[r].getName()){
                sign=true;
            }
            break;
        case 3:
            if(L[l].getE_mail()<=R[r].getE_mail()){
                sign=true;
            }
            break;
        case 4:
            if(L[l].getStart_time()<=R[r].getStart_time()){
                sign=true;
            }
            break;
        case 5:
            if(L[l].getEnd_time()<=R[r].getEnd_time()){
                sign=true;
            }
            break;
    }
    return sign;
}
template<class T>
void Merge_sort_recursive(vector<T> &A,int p,int r,int choice){
    if(p<r){
        int q=(p+r)/2;
        Merge_sort_recursive(A,p,q,choice);
        Merge_sort_recursive(A,q+1,r,choice);
        Merge(A,p,q,r,choice);
    }
}
template<class T>
void Merge_pass(vector<T> &A,int N, int length,int choice)
{ /* 两两归并相邻有序子列 */
    int i, j;

```

```

        for ( i=0; i <= N-2*length; i += 2*length ){
            Merge( A, i, i+length-1, i+2*length-1,choice);
        }
        if ( i+length < N ){
            Merge( A, i, i+length-1, N-1,choice);
        } /* 归并最后 2 个子列*/
    }

    template<class T>
    void Merge_sort_non_recursive(vector<T> &A, int N,int choice)
    {
        int length;
        length = 1;
        while( length < N ) {
            Merge_pass( A,N,length,choice);
            length *= 2;
        }
    }

    int string_to_int(string str){
        stringstream ss;
        int s;
        ss<<str;
        ss>>s;
        return s;
    }//将字符串转换为整数

    void Write_file_solution(string filename,vector<Advertisement> &advertisments){
        int order=1;
        ofstream of(filename);
        for(int i=0;i<advertisments.size();i++){
            of<<"Ad Order:"<<order++<<endl;
            of<<"Ad id:"<<advertisments[i].getId()<<endl;
            of<<"Ad amount:"<<advertisments[i].getAmount()<<endl;
            of<<"Ad name:"<<advertisments[i].getName()<<endl;
            of<<"Ad email:"<<advertisments[i].getEmail()<<endl;
            of<<"Ad start_time:"<<advertisments[i].getStart_time()<<endl;
            of<<"Ad end_time:"<<advertisments[i].getEnd_time()<<endl;
        }
        of.close();
    }

    //vector<int> &ids,vector<string> &names,vector<int> amounts,vector<string> emails,
    void Open_file_solution(string filename,vector<Advertisement> &advertisments){
        int i=0;
        int id=1;
        int res;
        vector<int> ids;

```

```

vector<string> names;
vector<int> amounts;
vector<string> emails;
vector<string> starttime;
vector<string> endtime;
ifstream in_file;
in_file.open(filename);
string item;
getline(in_file,item);
cout<<"The input content in the input.txt file is as follows:"<<endl;
while(in_file){
    i++;
    res=i%5;
    cout<<i<<" : "<<item<<endl;
    switch(res){
        case 1:
            names.push_back(item);
            break;
        case 2:
            amounts.push_back(string_to_int(item));
            break;
        case 3:
            emails.push_back(item);
            break;
        case 4:
            starttime.push_back(item);
            break;
        case 0:
            endtime.push_back(item);
            break;
        default:
            break;
    }
    getline(in_file,item);
}
int index=0;
for(int index=0;index<names.size();index++){
    Advertisement
    ad(id,amounts[index],names[index],emails[index],starttime[index],endtime[index]);
    advertisements.push_back(ad);
    id++;
}
in_file.close();
}

```

```

};

int main(){
    Utils m;
    vector<Advertisement> ads;
    m.Open_file_solution("L://LAB/input.txt",ads);
    int choice;
    cout<<endl;
    cout<<"*****" << endl;
    cout<<"please input the way you want to sort the advertisements:\nThe Advertisement has the
following attributes:<<endl;
    cout<<"amount,name,email,start_time,end_time"<<endl;
    cout<<endl;
    cout<<"%%%%%%%%%%%%% Welcome to the advertising system %%%%%%%%%%"<<endl;
    cout<<"***** 0 for exit *****" << endl;
    cout<<"***** 1 for amount *****" << endl;
    cout<<"***** 2 for name *****" << endl;
    cout<<"***** 3 for email *****" << endl;
    cout<<"***** 4 for start_time *****" << endl;
    cout<<"***** 5 for end_time *****" << endl;
    cout<<"%%%%%%%%%%%%%" << endl;
    cout<<"please input your choice: ";
    cin>>choice;
    if(choice!=0){
        m.Merge_sort_recursive(ads,0,ads.size()-1,choice);
        m.Write_file_solution("L://LAB/output.txt",ads);
        cout<<"please observe the results in the txt file: output.txt"<<endl;
    }else{
        cout<<"Exit the system successfully!"<<endl;
    }
    system("pause");
}

```

### 实验 3.3

```

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

int* X = new int[1024];
int* Y = new int[1024];

```

```
//读取文件
void Open_File(string filename, int& n, int* X, int* Y) {
    ifstream infile;
    infile.open(filename);
    if (!infile.is_open())//判断文件是否成功打开
    {
        cout << "文件没成功打开" << endl;
        exit(1);
    }
    string str;
    getline(infile,str);
    n = stoi(str);
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            getline(infile, str); //获取数组中的最后一个数据
            X[i] = stoi(str);
        }else{
            getline(infile, str, ' '); //获取数组中的数据，由空格间隔
            X[i]= stoi(str);
        }
    }
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            getline(infile, str);
            Y[i] = stoi(str);
        }
        else {
            getline(infile, str, ' ');
            Y[i] = stoi(str);
        }
    }
    infile.close();
}

//写入文件
void write_File(string filename,double res) {
    ofstream outfile;
    outfile.open(filename);
    if (!outfile.is_open())//判断文件是否成功打开
    {
        cout << "文件没成功打开" << endl;
        exit(1);
    }
    outfile << res << endl;
```

```

}

double Get_Median(int* A, int n) {
    double m = 0;
    if (n % 2 == 0) { //若数组中数据个数为偶数
        m = (A[n / 2 - 1] + A[n / 2]) / 2.0;
    }
    else { //若数组中数据个数为奇数
        m = A[n / 2];
    }
    return m; //返回浮点数类型
}

double Count_Median(int* X, int* Y, int n) { //计算两个有序数组的中位数
    double x = Get_Median(X, n);
    double y = Get_Median(Y, n);
    double z = 0; //z 为最终两个有序数组的中位数
    if (n == 1) { //直接计算两个数的平均值为中位数
        z = (x + y) / 2;
        return z;
    }
    else if (n == 2) {
        z = (max(X[0], Y[0]) + min(X[1], Y[1])) / 2;
        return z;
    }
    if (x == y) {
        z = x;
        return z;
    }
    else if (x < y) { //舍去 X 中小于 x 的所有前半部分数据以及 Y 中大于 y 的所有后半部分数据, //得到新的数组 X1,Y1 和数组大小 n1
        if (n % 2 == 0) { //当 n 为偶数时
            //保留 X 中最后一个小于 x 的数和 Y 中第一个大于 y 的数, 即保留原来计算中位数的两个数
            int* X1 = X + n / 2 - 1; //数组 X1 取 X 的下标为 n/2-1 的数及其之后所有数
            int* Y1 = Y; //数组 Y1 取 Y 中下标为 0 直到下标为 n/2 的数
            int n1 = n - n / 2 + 1;
            return Count_Median(X1, Y1, n1);
        }
        else {
            int* X1 = X + (n - 1) / 2; //数组 X1 取 X 的下标为(n-1)/2 的数及其之后所有数
            int* Y1 = Y; //数组 Y1 取 Y 中下标为 0 直到下标为(n-1)/2-1 的数
            int n1 = n - (n - 1) / 2;
            return Count_Median(X1, Y1, n1);
        }
    }
    else if (x > y) { //操作思路和上述相同
        if (n % 2 == 0) {
            int* X2 = X;
            int* Y2 = Y + n / 2 - 1;
            int n2 = n - n / 2 + 1;
        }
    }
}

```

```
    return Count_Median(X2, Y2, n2);
}else {
    int* X2 = X;
    int* Y2 = Y + (n - 1) / 2;
    int n2 = n - (n - 1) / 2;
    return Count_Median(X2, Y2, n2);
}
}
return -1;
}
int main() {
    int n;
    string infilename, outfilename;
    infilename = "L://LAB/input_lab3.1.txt";
    outfilename = "L://LAB/output_lab3.1.txt";
    Open_File(infilename, n, X, Y);
    double res = Count_Median(X, Y, n);
    write_File(outfilename, res);
    cout<<"The result is: "<<res<<endl;
    system("pause");
}
```

# 实验报告

课程名称：数据结构与算法分析 班级：软信 2001 实验成绩：  
实验名称：动态规划 学号：20207130 批阅教师签字：  
实验编号：实验四 姓名：赖骏鸿 实验日期：2022 年 6 月 3 日  
指导教师：马毅 组号：暂无 实验时间：12 时 44 分— 16 时 10 分

## 一、实验目的

- (1) 熟练掌握动态规划思想及教材中相关经典算法。
- (2) 掌握用动态规划解题的基本步骤，能够用动态规划解决一些问题。

## 二、实验内容

本实验选择了找零钱问题（难度系数为 3，记作实验 4.1），租用游艇问题（难度系数为 4，记作实验 4.2）

### 1. 找零钱问题（难度系数为 3，记作实验 4.1）

#### ★ 问题描述

设有  $n$  种不同面值的硬币，各硬币的面值存于数组  $T[1:n]$  中。现要用这些面值的硬币来找钱，可以实用的各种面值的硬币个数不限。当只用硬币面值  $T[1], T[2], \dots, T[i]$  时，可找出钱数  $j$  的最少硬币个数记为  $C(i, j)$ 。若只用这些硬币面值，找不出钱数  $j$  时，记  $C(i, j)=\infty$ 。

#### ★ 编程任务

设计一个动态规划算法，对  $1 \leq j \leq L$ ，计算出所有的  $C(n, j)$ 。算法中只允许实用一个长度为  $L$  的数组。用  $L$  和  $n$  作为变量来表示算法的计算时间复杂性

#### ★ 数据输入

由文件 input.txt 提供输入数据。文件的第 1 行中有 1 个正整数  $n$  ( $n \leq 13$ )，表示有  $n$  种硬币可选。接下来的一行是每种硬币的面值。由用户输入待找钱数  $j$ 。

#### ★ 结果输出

程序运行结束时，将计算出的所需最少硬币个数输出到文件 output.txt 中。

#### 输入文件示例

```
input.txt
3
1 2 5
9
```

#### 输出文件示例

```
output.txt
3
```

#### ★ 实现提示

首先要建立递归关系，并将分析过程写在报告中。

### 2. 租用游艇问题（难度系数为 4，记作实验 4.2）

#### ★ 问题描述

长江游艇俱乐部在长江上设置了  $n$  个游艇出租站  $1, 2, \dots, n$ 。游客可在这些游艇出租站租用游艇，并在下游的任何一个游艇出租站归还游艇。游艇出租站  $i$  到游艇出租站  $j$  之间的租金为  $r(i, j)$ ,  $1 \leq i < j \leq n$ 。试设计一个算法，计算出从游艇

出租站 1 到游艇出租站 n 所需的最少租金。

### ★ 编程任务

对于给定的游艇出租站 i 到游艇出租站 j 之间的租金为  $r(i, j)$ ,  $1 \leq i < j \leq n$ , 编程计算从游艇出租站 1 到游艇出租站 n 所需的最少租金。

### ★ 数据输入

由文件 input.txt 提供输入数据。文件的第 1 行中有 1 个正整数 n ( $n \leq 200$ ) , 表示有 n 个游艇出租站。接下来的  $n-1$  行是  $r(i, j)$ ,  $1 \leq i < j \leq n$ 。

### ★ 结果输出

程序运行结束时, 将计算出的从游艇出租站 1 到游艇出租站 n 所需的最少租金输出到文件 output.txt 中。

输入文件示例	输出文件示例
input.txt	output.txt
3	12
5 15	
7	

### ★ 实现提示

建立递归关系, 然后按照递归关系写出算法。

## 三、实验环境

实验环境:

操作系统:

windows10

调试软件名称及版本号:

Visual Studio Code February 2022 (version 1.65)

编程语言及版本号:

C++/C

上机地点:

班级未集中进行实验课

机器台号:

华硕飞行堡垒 8

## 四、实验过程与问题分析

(1) 对问题的简单分析。

答: 对于找零钱的问题(**实验 4.1**), 我们采用动态规划的思想来求解。此问题类似于 01 背包问题。01 背包问题是这样描述的: 给定 n 种物品和一个背包。物品 i 的重量为  $w_i > 0$ , 其价值为  $v_i > 0$ , 背包的容量为 c, 求一组数  $x_i$  ( $x_i = 0$  或 1,  $x_i = 0$  表示物体 i 不放入背包,  $x_i = 1$  表示把物体 i 放入背包), 使得下式最大:

$$\max \sum_{i=1}^n v_i x_i$$

即在书包容量一定的情况下，将尽可能多的价值装入背包。给定  $c > 0, w_i > 0, v_i > 0, 1 \leq i \leq n$ , 我们使用 **knap(1,n,c)** 来表示满足以下约束条件：

$$s.t \left\{ \begin{array}{l} \max = \sum_i^n v_i x_i \\ \sum_i^n w_i x_i \leq c \\ x_i \in \{0, 1\}, i = 1, 2, \dots, n \end{array} \right.$$

事实上，这是一个最优化问题，因而可以考虑选择动态规划来解决。下面证明其具有最优子结构性质：即若背包容量为  $c$  时， $(y_1, y_2, \dots, y_n)$  为待选物品从 1 到  $n$  时 **knap(1,n,c)** 的最优解，则 $(y_2, \dots, y_n)$ 将是物品 1 做出选择后的子问题 **knap(2,n,c-w1\*y1)**的最优解。当第一个物品做出决策后，有两种状态：① $y_1=0$ ，则背包容量没有影响， $(y_2, \dots, y_n)$ 将是物品 1 做出选择后的子问题 **knap(2,n,c)**的最优解；② $y_1=1$ ，则背包减少  $w_1$ ，价值增长  $v_1$ ， $(y_2, \dots, y_n)$ 将是物品 1 做出选择后的子问题 **knap(2,n,c-w1)**的最优解。很容易我们可以得到并证明最优解具有唯一性。接下来我们梳理背包问题的递归关系：设背包问题的子问题的最优值为  $m(i,j)$ ，即  $m(i,j)$  是背包容量为  $j$ ，可选择物品为  $i, i+1, \dots, n$  的最优值。

---当选择第  $i$  个物品时， $m(i,j)=m(i+1,j-w_i)+v_i$ ；

---当不选择第  $i$  个物品时， $m(i,j)=m(i+1,j)$ ；

由 01 背包问题的最优子结构性质，我们可以建立计算  $m(i,j)$  的递归式如下：

$$m(i,j) = \begin{cases} x = \max\{m(i+1,j), m(i+1,j-w_i+v_i)\}, \text{while } j \geq w_i \\ x = m(i+1,j), \text{while } 0 \leq j < w_i \end{cases}$$

$$m(n,j) = \begin{cases} v_n & j \geq w_n \\ 0 & 0 \leq j < w_n \end{cases}$$

类似的我们可以依据上文来分析**找零钱问题(实验 4.1)**：设有  $n$  种不同面值的硬币，各硬币的面值存于数组  $T[1:n]$  中。现要用这些面值的硬币来找钱，可以实用的各种面值的硬币个数不限。当只用硬币面值  $T[1], T[2], \dots, T[i]$  时，可找出钱数  $j$  的最少硬币个数记为  $C(i, j)$ 。若只用这些硬币面值，找不出钱数  $j$  时，记  $C(i, j)=\infty$ 。即在给定钱数  $j$  的情况下，找到对应的  $x_i$ ，使得下式成立：

$$s.t \left\{ \begin{array}{l} j = \sum_i^n T_i x_i \\ \min \sum_i^n x_i \\ x_i \in N, i = 1, 2, \dots, n \end{array} \right.$$

即在给定钱数的情况下，找到所需钱数最小的数量。事实上，这是一个最优化问题，因而可

以考虑选择动态规划来解决。下面证明其具有最优子结构性质：即目标钱数为 target 时， $(y_1, y_2, \dots, y_n)$  为待选硬币从 1 到 n 时  $\text{charge}(1, n, \text{target})$  的最优解，则 $(y_1, \dots, y_{n-1})$  将是硬币 n 做出支出后的子问题  $\text{charge}(1, n-1, \text{target} - y_n * T_n)$  的最优解。当第 n 个物品做出决策后，有两种状态：①  $y_n = 0$ ，则目标钱数没有影响， $(y_1, \dots, y_{n-1})$  将是物品 1 做出选择后的子问题  $\text{charge}(1, n-1, \text{target})$  的最优解；②  $y_n \neq 0$ ，则目标钱数减少  $y_n * T_n$ ， $(y_1, \dots, y_{n-1})$  将是硬币 n 做出选择后的子问题  $\text{charge}(1, n-1, \text{target} - y_n * T_n)$  的最优解。很容易我们可以得到并证明最优解具有唯一性。接下来我们梳理找零钱问题的递归关系：设找零钱问题的子问题的最优值为  $dp(i, j)$ ，即  $dp(i, j)$  是目标钱数为 j，可选择硬币为 0, 1, ..., i 的最优值。

---当选择了 k 个第 i 个硬币时， $dp(i, j) = dp(i-1, j-k*T_i) + k$ ；

---当因为目标钱数小于货币面值而不选择第 i 个硬币时， $dp(i, j) = dp(i-1, j)$

由找零钱问题的最优子结构性质，我们可以建立计算  $dp(i, j)$  的递归式如下：

$$dp(i, j) = \min_k \sum_{k=0}^{top} [dp(i-1, j - k * T_i) + k]$$

很容易看出，此问题的时间复杂度应该为  $O(n^3)$  的量级。

由此我们简单分析完了找零钱问题的动态规划基本结构。但是在依据这个思路完成本实验的时候，发现会出现很多难题与问题需要解决，算法思路比较复杂。参考了书上其他的思路，认为本算法的时空复杂度均可以进行优化，因而又采用了一个新的算法，新算法在不同程度上均有一定的优化：

#### 实验 4.1 新：

设货币面值集合 V 中有表示不同面值货币的元素  $v_1, v_2, v_3, \dots, v_n$ ，获得状态转换方程为：

$$\begin{cases} res[\text{target}] = \min_{0 < i \leq n} res[\text{target} - v_i] \\ res[j] = 0 \text{ while } j == 0 \\ res[j] = INT\_MAX - \varepsilon \text{ while } j != 0 \end{cases}$$

下面证明在此递归方程的条件下的最优子结构：

设需要换取的总金额为 target，货币的面值分别为  $v_1, v_2, v_3, \dots, v_k$ ，满足换取 y 的最少货币数对应到每种面值货币的张数为  $n_1, n_2, n_3, \dots, n_k$ 。此时换取的总张数为  $n_1 + n_2 + n_3 + \dots + n_k$ ，换取的公式为：

$$target = \sum_i^k v_i n_i$$

假设从这些货币中拿掉一张面值为  $v_1$  的货币，此时换取的总张数为  $n_1 + n_2 + n_3 + \dots + n_k - 1$ ，换算公式为：

$$target - v_1 = \left( \sum_{i=2}^k v_i n_i \right) + v_1(n_1 - 1)$$

假设此时换取  $target - v_1$  的总张数  $n_1 + n_2 + n_3 + \dots + n_{k-1}$  不是最少张数，则必然存在另一种换算方式为每种面值货币的张数对应  $m_1, m_2, m_3, \dots, m_k$ ，也就是总张数为  $m_1 + m_2 + m_3 + \dots + m_k$  张。进而推出换算  $target$  的最少张数为  $m_1 + m_2 + m_3 + \dots + m_k + 1$  张。然而已知总张数为  $n_1 + n_2 + n_3 + \dots + n_k$  为换取  $target$  的最少张数，不可能存在换算方式的总张数比这种方法还要少，产生了矛盾，因此找零钱问题满足最优子结构。由找零钱问题的最优子结构性质，我们可以建立计算  $res$  的递归式如下：

$$res[target] = \min_{0 < i \leq n} res[target - v_i]$$

显然此算法的时空复杂度均有提升，时间复杂度由  $O(n^3)$  数量级下降为  $O(n^2)$ ，空间复杂度由  $O(n^2)$  数量级下降为  $O(n)$ 。是一个不错的提升，算法效率大大提高。

对于租用游艇问题(实验 4.2)，题目要求对于给定的游艇出租站  $i$  到游艇出租站  $j$  之间的租金为  $r(i,j), 1 \leq i < j \leq n$ ，计算从游艇出租站 1 到游艇出租站  $n$  所需的最少租金。用数学语言来描述即为：

$$s.t \begin{cases} \min[r(1, n)] \\ \text{for: } r(i, j), 1 \leq i < j \leq n \end{cases}$$

事实上，这是一个最优化问题，因而可以考虑选择动态规划来解决。下面证明其具有最优子结构性质：即若  $r(i,j)$  为游艇出租站  $i$  到游艇出租站  $j$  之间最小的租金，那么对于任意的  $k$ ， $k$  满足  $i \leq k \leq j$ ， $r(i,k)$  和  $r(k,j)$  分别为游艇出租站  $i$  到游艇出租站  $k$  和游艇出租站  $k$  到游艇出租站  $j$  之间最小的租金。因此满足最优子结构性质。下面我们来梳理此问题的递归关系：

由问题的最优子结构性质，我们可以建立计算  $r(i,j)$  的递归表达式：

$$r(i, j) = \min \sum_k^{top} (r(i, k) + r(k, j))$$

很容易看出，此问题的时间复杂度应该为  $O(n^3)$  的量级。

由此我们简单分析完了租用游艇问题的动态规划基本结构。

(2) 根据对问题的分析，写出解决方法。

答：在**实验 4.1** 中，本程序中数据结构的设定与题目有所不同，假设 money[] 为面值递增存放顺序的 length 种钱币，所需要找的钱数为 target, dp[i][j] 表示对于 money[] 中的前 i 种硬币面值条件下目标钱数为 j 的最小的硬币数量。若输出的 dp[i][j] 绝对值大小为 INT\_MAX，那么表示在 j 的目标钱数下和 money[] 中的前 i 种硬币面值条件下，无法找到可以满足条件的硬币数量，即找不到可以兑换的组合数。采用自底向上的递归动态规划逐步求解即可。

在**实验 4.2** 中，此问题的求解相对于**实验 4.1** 要容易一点。对于求解 r(i,j) 的问题，我们将其划分为求解 r(i,k) 和 r(k,j) 的子问题。声明一个二维数组 dp, dp[i][j] 表示从游艇出租站 i 到游艇出租站 j 之间最小的租金。采用动态规划求解即可。

(3) 描述你在进行实现时，主要的函数或操作内部的主要算法：分析算法的时空复杂度，并说明设计的巧妙之处，如有创新，请清晰的表述。

答：在**实验 4.1** 中：

主函数：

```
int main(){
    vector<string> store;
    string filename="L://LAB/input_lab4.1.txt";
    string outfilename="L://LAB/output_lab4.1.txt";
    int money_number;
    vector<int> money;
    int target;
    store=Open_file(filename,store);
    for(int i=0;i<store.size();i++){
        if(i==0){
            money_number=stoi(store[i]);
        }else if(i==store.size()-1){
            target=stoi(store[i]);
        }else{
            money.push_back(stoi(store[i]));
        }
    }
    if(money[0]!=1){
        money.insert(money.begin(),1);
    }
    Write_file(outfilename,solution(money,target));
    cout<<"The final result of the dynamic programming calculation is:"<<endl;
    correct(money,target);
    Print_dp(money,target);
    system("pause");
}
```

主要函数：

函数	说明
int Min(int a,int b)	返回 a 和 b 中的较小值
int solution(vector<int> &money,int target)	找零钱问题的主逻辑函数
void Print_dp(vector<int> &money,int target)	打印动态规划求解过程

---

```
template<class T>
vector<T>&Open_file(string filename,vector<T> &store) 从文件中读取有效数据
```

```
void Write_file(string filename,int res)
```

将结果写在文件中

```
int solution2(vector<int> &money,int target)
```

优化后的动态规划算法

---

1) vector<T>&Open\_file

```
template<class T>
vector<T> &Open_file(string filename,vector<T> &store){
    ifstream in_file;
    in_file.open(filename);
    string item;
    getline(in_file,item);
    int length=stoi(item);
    store.push_back(item);
    for(int i=0;i<length;i++){
        if(i==length-1){
            getline(in_file,item);
        }else{
            getline(in_file,item,' ');
        }
        store.push_back(item);
    }
    getline(in_file,item);
    store.push_back(item);
    return store;
}
```

此函数主要完成从文件 filename 中读取数据并存储在 vector 数组 store 中的工作。

时间复杂度:  $O(n)$ , $n$  为数据规模大小, 即不同面值硬币的种类

空间复杂度:  $O(1)$

2) int solution

```

int solution(vector<int> &money, int target){
    for(int i=0;i<money.size();i++){
        dp[i][0]=0;
    }
    for(int j=0;j<=target;j++){
        if(j%money[0]==0){
            dp[0][j]=j/money[0];
            // dp[0][j]=1;
        }else{
            dp[0][j]=(int)ceil((double)j/money[0]);
        }
    }
    for(int i=1;i<money.size();i++){
        for(int j=1;j<=target;j++){
            int temp=INT_MAX;
            for(int k=0;j-k*money[i]>=0;k++){
                temp=min(dp[i-1][j-k*money[i]]+k,temp);
                if(j==k*money[i]||test[i-1][j-k*money[i]]==1){
                    test[i][j]=1;
                }
            }
            dp[i][j]=temp;
        }
    }
    return dp[money.size()-1][target];
}

```

此函数为求解找零钱问题的主逻辑函数，在这里进行详细的解释。这里声明的变量名含义如下：money 表示不同面值的硬币数组，target 表示目标钱数，dp[i][j] 表示对于 money[] 中的前 i 种硬币面值条件下目标钱数为 j 的最小的硬币数量。这里为了方便我们声明 dp 数组大小为 dp[1024][1024]，标志位数组 test[1024][1024]。

```

int dp[1024][1024]={0};
int ddpp[1024][1024]={0};
int test[1024][1024]={0};

```

首先对于：

```

for(int i=0;i<money.size();i++){
    dp[i][0]=0;
}

```

表示对于目标钱数为 0 的情况下，对于任何不同面值硬币的组合都仅仅能找到 0 种方法，即在不同面值硬币的数量全部为 0 的情况下目标钱数才可能为 0。对于：

```

for(int j=0;j<=target;j++){
    if(j%money[0]==0){
        dp[0][j]=j/money[0];
    }else{
        dp[0][j]=(int)ceil((double)j/money[0]);
        // dp[0][j]=INT_MAX;
        // dp[0][j]=ceil(j/money[0]);
    }
}

```

表示对于第一个面值的硬币下即 `money[0]`, 对于 0 到 `target` 等不同目标钱数情况下, 对应目标钱数 `j` 所需要第一个面值的硬币下即 `money[0]` 的数量。如果 `j` 恰好可以被 `money[0]` 整除, 那么令 `dp[0][j]` 等于它们的商; 否则则表示钱数 `j` 在面值 `money[0]` 下找不到一个正整数 `p` 满足 `p*money[0]=j`, 那么令 `dp[0][j]` 等于他们商的向上取整值。但是为了解决问题的方便, 我们在 `vector` 数组 `money` 传进此函数之前做了一些处理:

此处为该算法的设计巧妙之处, 判断 `money` 的第一个元素是否为 1。为什么这样判断呢? 因为对于递归方程:

$$dp(i, j) = \min_k \sum_{k=1}^{top} [dp(i-1, j - k * T_i) + k]$$

我们可以知道对于求解 `dp[i][j]`, 必须依赖于 `dp[i-1][j']`, 其中 `j'<=j`, 那么本质上对于此问题的求解依赖于 `dp[0][j']` 的所有值, 因此对于数组 `dp` 的提前正确的初始化非常重要。而为了方便数据处理, 因为 1 是所有数的约数, 因而采取提前初始化面值为 1 的 `dp` 数据。

接下来是循环求解的过程:

```

for(int i=1;i<money.size();i++){
    for(int j=1;j<=target;j++){
        int temp=INT_MAX;
        for(int k=0;j-k*money[i]>=0;k++){
            temp=Min(dp[i-1][j-k*money[i]]+k,temp);
            if(j==k*money[i]||test[i-1][j-k*money[i]]==1){
                test[i][j]=1;
            }
        }
        dp[i][j]=temp;
    }
}

```

每次进入最内侧循环时, 需要判断对应的标志位对于全局是否合法。此处合法与否的意思是判断前面加入的面值为 1 的硬币是否真正会影响到对于目标钱数 `j` 的最小硬币的取值。这里

我们举个例子，对于下列的 dp 数组，正常情况下为：

硬币面值\目标钱数	0	1	2	3	4	5	6	7	8	9	10	11
3	0	-1	-1	1	-1	-1	2	-1	-1	3	-1	-1
4	0	-1	-1	-1	1	-1	2	2	2	3	3	3

但是我们为了便于算法后续的简单处理，提前加入了面值为 1 的硬币，那么 dp 数组变为：

硬币面值\目标钱数	0	1	2	3	4	5	6	7	8	9	10	11
1	0	1	2	3	4	5	6	7	8	9	10	11
3	0	1	2	1	2	3	2	3	4	3	4	5
4	0	1	2	3	1	2	2	2	2	3	3	3

即虽然加入面值为 1 的硬币便于了我们数据的处理，但是完全打乱了我们的求解，通过仔细 debug 发现规律：若后续求解使用到了  $dp[0][1:target+1]$  的值，那么就判定为不合法；反过来想，如果后续求解最小值过程中仅仅调用了  $dp[0][0]$ ,或者调用了仅仅调用  $dp[0][0]$  的数据，那么就是合法的数据。例如对于  $dp[2][10]$ ，实际上  $dp[2][10]=dp[1][6]+1=3$ , 而  $dp[1][6]=dp[0][0]+2$ , 因而数据  $dp[1][6]$  和  $dp[2][10]$  都是合法数据；而对于  $dp[2][5]=dp[1][1]+1, dp[1][1]=dp[0][1]+0$ , 数据  $dp[1][1]$  和  $dp[2][5]$  均不是合法数据，因而在后续中对 dp 数组会进行修正，调用函数：

```
void correct(vector<int> &money, int target){
    for(int i=0;i<money.size();i++){
        for(int j=1;j<=target;j++){
            if(test[i][j]==0){
                dp[i][j]=-1;
            }
        }
    }
}
```

由于前面我们已经对列数和行数均为 0 的值进行了初始化，因此这里的循环索引值均从 1 开始。找到问题分析(1)中的递归式子如下：

$$dp(i, j) = \min_k \sum_{k=1}^{top} [dp(i-1, j-k*money[i]) + k]$$

很显然，对于此问题的求解需要 3 个循环。在每次进入第三层循环之前（循环索引为 k），声明一个 temp 变量为无穷大，然后当  $j-k*money[i]$  大于等于 0 的条件下，求出 temp 值和  $dp[i-1][j-k*money[i]]$  中较小的值，并将值赋值给 temp,继续循环。最终的 temp 值就是上述递归式的一个子问题的解，赋值给  $dp[i][j]$  即可。

最终，返回硬币数量最小的情况：

```
    return dp[money.size()-1][target];
```

时间复杂度:  $O(m \cdot n^2)$ ,  $m$  为不同面值硬币种类的数量,  $n$  为目标钱数的大小。对于问题的求解有三层循环, 第一层循环为  $i$  从 1 到  $m$ , 第二层循环是  $j$  从 1 到  $n$ , 第三层循环是  $k$  从 0 到  $j/money[i]$ , 循环数量级与  $j$  相同即为  $n$ 。因而本算法时间复杂度为  $O(m \cdot n^2)$ 。

空间复杂度:  $O(m \cdot n)$ ,  $m$  为不同面值硬币种类的数量,  $n$  为目标钱数的大小。本算法提前申请了二维数组  $dp$ , 其有效利用大小为  $dp[m][n+1]$ , 因此空间复杂度为  $O(m \cdot n)$ 。

3) int solution2(vector<int> &money, int target)

```
int solution2(vector<int> &money, int target){
    for(int i=1;i<=target;i++){
        res[i]=INT_MAX-10000;
    }
    for(int i=1;i<=target;i++){
        for(int j=0;j<money.size();j++){
            int temp=INT_MAX;
            if(i-money[j]>=0){
                temp=min(res[i-money[j]]+1,res[i]);
                res[i]=temp;
            }
        }
    }
    return res[target];
}
```

此函数是本题目的改进算法。首先初始化一维数组  $res$ , 除了  $res[0]=0$ ,  $res$  其他的值都是一个较大的数, 这里赋值为  $INT\_MAX-10000$ 。因为考虑到后续

```
temp=min(res[i-money[j]]+1,res[i]);
res[i]=temp;
```

的+1 操作的情况, 防止数发生越界。采用递归式的思想:

$$res[target] = \min_{0 < i \leq n} res[target - v_i]$$

将兑换硬币的大目标钱数问题转换成若干兑换硬币的小目标钱数的子问题, 逐步求解, 并且使用  $res$  数组存取数据即可。显然此算法的时空复杂度均有提升, 时间复杂度由  $O(n^3)$  数量级下降为  $O(n^2)$ , 空间复杂度由  $O(n^2)$  数量级下降为  $O(n)$ 。是一个不错的提升, 算法效率大大提高。

时间复杂度:  $O(n \cdot m)$ ,  $n$  为目标钱数的大小,  $m$  为面值种类的数量。

空间复杂度:  $O(n)$ ,  $n$  为目标钱数的大小。

在实验 4.2 中:

主函数:

```

int main(){
    vector<string> store;
    string filename="L://LAB/input_lab4.2.txt";
    string outfilename="L://LAB/output_lab4.2.txt";
    int length;
    vector<int> rent;
    store=Open_file(filename,store);
    for(int i=0;i<store.size();i++){
        if(i==0){
            length=stoi(store[i]);
        }else{
            rent.push_back(stoi(store[i]));
        }
    }
    write_file(outfilename,solution(rent,length));
    Print_dp(rent,length);
    system("pause");
}

```

主要函数：

函数	说明
<code>template&lt;class T&gt;</code> <code>vector&lt;T&gt; &amp;Open_file(string filename,vector&lt;T&gt; &amp;store)</code>	从文件中获取数据
<code>void Write_file(string filename,int res)</code>	将程序运行结果写到文件中
<code>int solution(vector&lt;int&gt; &amp;rent,int sum)</code>	租用游艇主逻辑算法
<code>void Print_dp(vector&lt;int&gt; &amp;rent,int sum)</code>	打印 dp 数组

1) `vector<T> &Open_file`

```
template<class T>
vector<T> &Open_file(string filename, vector<T> &store){
    ifstream in_file;
    in_file.open(filename);
    string item;
    getline(in_file, item);
    int length=stoi(item);
    store.push_back(item);
    for(int index=1;index<length;index++){
        for(int i=0;i<length-index;i++){
            if(i==length-index-1){
                |   getline(in_file, item);
            }else{
                |   getline(in_file, item, ' ');
            }
            store.push_back(item);
        }
    }
    getline(in_file, item);
    store.push_back(item);
    return store;
}
```

此函数主要完成从文件中读取文件的操作。

时间复杂度:  $O(n^2)$ , $n$  为读取到的游艇站的数量。

空间复杂度:  $O(1)$

2) int solution

```

int solution(vector<int> &rent,int sum){
    int index=0;
    for(int i=0;i<sum;i++){
        for(int j=i;j<sum;j++){
            if(i==j){
                dp[i][j]=0;
                continue;
            }else{
                dp[i][j]=rent[index];
                index++;
            }
        }
    }
    for(int i=0;i<sum;i++){
        for(int j=i;j<sum;j++){
            if(i==j){
                continue;
            }else{
                int temp=INT_MAX;
                for(int k=i;k<=j;k++){
                    temp=min(temp,dp[i][k]+dp[k][j]);
                }
                dp[i][j]=temp;
            }
        }
    }
    for(int i=0;i<sum;i++){
        for(int j=i;j<sum;j++){
            dp[j][i]=dp[i][j];
        }
    }
    return dp[0][sum-1];
}

```

此函数为运用动态规划求解问题的主逻辑算法。在这里进行详细的说明。首先提前声明了 dp 数组：

```
int dp[1024][1024];
```

函数入口参数 rent 表示存放租金的 vector 数组， sum 表示游艇站的数量。函数第一部分：

```

int index=0;
for(int i=0;i<sum;i++){
    for(int j=i;j<sum;j++){
        if(i==j){
            dp[i][j]=0;
            continue;
        }else{
            dp[i][j]=rent[index];
            index++;
        }
    }
}

```

表示对  $dp$  数组进行初始化，对角线的数组游艇出租站到自身的租金，显然为 0；对于非对角线的  $dp$  数组采用  $rent$  数组进行赋值。观察到根据数组  $dp$  的具体实际意义，由于  $dp$  数组是对称的，因而我们的所有操作均仅仅在数组（矩阵）的上半对角线数组（矩阵）上。函数的第二部分：

```

for(int i=0;i<sum;i++){
    for(int j=i;j<sum;j++){
        if(i==j){
            continue;
        }else{
            int temp=INT_MAX;
            for(int k=i;k<=j;k++){
                temp=min(temp,dp[i][k]+dp[k][j]);
            }
            dp[i][j]=temp;
        }
    }
}

```

此部分为动态规划过程。首先对于非对角线上的元素进行迭代，观察我们之前得到的递归方程：

$$r(i, j) = \min \sum_{k}^{top} (r(i, k) + r(k, j))$$

容易得到：

$$dp[i][j] = \min \sum_{k}^{top} (dp[i][k] + dp[k][j])$$

最内侧循环开始前，初始化  $temp$  为无穷大。最终得到最小的租金，然后对  $dp$  进行更新： $dp[i][j]=temp$ 。最后一部分：

```

for(int i=0;i<sum;i++){
    for(int j=i;j<sum;j++){
        |
        dp[j][i]=dp[i][j];
    }
}

```

补全 dp 数组。最后返回  $dp[0][sum-1]$  即表示租用游艇从头到尾的租金：

```
return dp[0][sum-1];
```

**时间复杂度：** $O(n^3)$ , $n$  为读取到的游艇站的数量。在动态规划的求解部分中采用了三层 for 循环，并且每一层 for 循环的数量级均为  $n$ 。因此时间复杂度为  $O(n^3)$ 。

**空间复杂度：** $O(n^2)$ , $n$  为读取到的游艇站的数量。申请了 dp 数组，数组的大小为  $n^2$ 。

(4) 你在调试过程中发现了什么样的问题？又作出了什么样的改进？

答：对于**实验 4.1**，我在初次编写算法的时候，忽略了题目一个条件，即“若只用这些硬币面值，找不出钱数  $j$  时，记  $C(i, j)=\infty$ 。”因此在数组初始化的时候写出了以下代码：

```

int solution(vector<int> &money, int target){
    for(int i=0;i<money.size();i++){
        |
        dp[i][0]=0;
    }
    for(int j=0;j<=target;j++){
        |
        if(j%money[0]==0){
            |
            dp[0][j]=j/money[0];
        }else{
            |
            dp[0][j]=ceil(j/money[0]);
            // dp[0][j]=INT_MAX;
        }
    }
    for(int i=1;i<money.size();i++){
        for(int j=1;j<=target;j++){
            |
            int temp=INT_MAX;
            for(int k=0;j-k*money[i]>=0;k++){
                |
                temp=min(dp[i-1][j-k*money[i]]+k,temp);
            }
            dp[i][j]=temp;
        }
    }
    return dp[money.size()-1][target];
}

```

在对二维数组 dp 中第一个面值  $money[0]$  对应行的数据初始化中，表示对于第一个面值的硬币下即  $money[0]$ ，对于 0 到  $target$  等不同目标钱数情况下，对应目标钱数  $j$  所需要第一个面值的硬币下即  $money[0]$  的数量。如果  $j$  恰好可以被  $money[0]$  整除，那么令  $dp[0][j]$  等于它们的商；否则则表示钱数  $j$  在面值  $money[0]$  下找不到一个正整数  $p$  满足  $p*money[0]=j$ ，那么令  $dp[0][j]$  等于它们的商向上取整。这会导致什么问题呢？所有的目标钱数在任意的不同面值组合下均会找到一个合法的数量，这显然是不符合题意的。例如在 input.txt 中的数据如下：

\*input\_lab4.1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
2
2 3
7
```

程序运行结果：

```
0yuan    1yuan    2yuan    3yuan    4yuan    5yuan    6yuan    7yuan
2yuan: 0 2yuan: 0 2yuan: 1 2yuan: 1 2yuan: 2 2yuan: 2 2yuan: 3 2yuan: 3
3yuan: 0 3yuan: 0 3yuan: 1 3yuan: 1 3yuan: 1 3yuan: 2 3yuan: 2 3yuan: 2
请按任意键继续. . .
```

简单来看，对于目标钱数为 7，面值为 2 的情况下应该是至少需要 4 张的，但是程序会显示最少需要 3 张，这显然是不符合题意的。于是将代码进一步修改为：

```
for(int j=0;j<=target;j++){
    if(j%money[0]==0){
        dp[0][j]=j/money[0];
    }else{
        dp[0][j]=(int)ceil((double)j/money[0]);
        // dp[0][j]=INT_MAX;
        // dp[0][j]=0;
    }
}
```

此代码下的运行结果：

```
0yuan    1yuan    2yuan    3yuan    4yuan    5yuan    6yuan    7yuan
2yuan: 0 2yuan: 1 2yuan: 1 2yuan: 2 2yuan: 2 2yuan: 3 2yuan: 3 2yuan: 4
3yuan: 0 3yuan: 1 3yuan: 1 3yuan: 1 3yuan: 2 3yuan: 2 3yuan: 2 3yuan: 3
请按任意键继续. . .
```

显然此时仍然不符合题意。在后续的改进中发现有以下主要问题，对于递归方程：

$$dp(i, j) = \min_k \sum_{k=1}^{top} [dp(i-1, j-k*T_i) + k]$$

我们可以知道对于求解  $dp[i][j]$ , 必须依赖于  $dp[i-1][j']$ , 其中  $j' \leq j$ , 那么本质上对于此问题的求解依赖于  $dp[0][j']$  的所有值，因此对于数组  $dp$  的提前正确的初始化非常重要。但是在实验 debug 过程中，发现自己的程序仅仅能算出对应的目标钱数所需要的最小的硬币数量，却无法判断对应目标钱数是否能找到。因此做出如下改进：

首先声明标志位数组  $test[1024][1024]$ 。

①主函数中添加如下代码：

```

if(money[0]!=1){
    money.insert(money.begin(),1);
}

```

②增添修正函数：

```

void correct(vector<int> &money,int target){
    for(int i=0;i<money.size();i++){
        for(int j=1;j<=target;j++){
            if(test[i][j]==0){
                dp[i][j]=-1;
            }
        }
    }
}

```

③在合理条件下修改标志位数组：

```

for(int i=1;i<money.size();i++){
    for(int j=1;j<=target;j++){
        int temp=INT_MAX;
        for(int k=0;j-k*money[i]>=0;k++){
            temp=min(dp[i-1][j-k*money[i]]+k,temp);
            if(j==k*money[i]||test[i-1][j-k*money[i]]==1){
                test[i][j]=1;
            }
        }
        dp[i][j]=temp;
    }
}

```

最终得到的运行结果为：

```

c:\VS\code\c and c++\datastructure\lab4.1.exe
The final result of the dynamic programming calculation is:
0yuan 1yuan 2yuan 3yuan 4yuan 5yuan 6yuan 7yuan
1yuan: 0 1yuan: -1 1yuan: -1 1yuan: -1 1yuan: -1 1yuan: -1 1yuan: -1
2yuan: 0 2yuan: -1 2yuan: 1 2yuan: -1 2yuan: 2 2yuan: -1 2yuan: 3 2yuan: -1
3yuan: 0 3yuan: -1 3yuan: 1 3yuan: 1 3yuan: 2 3yuan: 2 3yuan: 3
请按任意键继续. . .

```

其中-1 表示无法找到对应的零钱。

对于**实验 4.2**，暂时未发现问题。

## 五、实验结果总结

回答以下问题：

(1)你的测试充分吗？为什么？你是怎样考虑的？

答：对于**实验 4.1**，测试结果如下：

测试一：

输入:

input\_lab4.1.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
3  
1 2 5  
9

程序运行结果:

```
c:\VS\code\c and c++\datastructure\lab4.1.exe  
The final result of the dynamic programming calculation is:  
0yuan 1yuan 2yuan 3yuan 4yuan 5yuan 6yuan 7yuan 8yuan 9yuan  
lyuan: 0 lyuan: 1 lyuan: 2 lyuan: 3 lyuan: 4 lyuan: 5 lyuan: 6 lyuan: 7 lyuan: 8 lyuan: 9  
2yuan: 0 2yuan: 1 2yuan: 1 2yuan: 2 2yuan: 2 2yuan: 3 2yuan: 3 2yuan: 4 2yuan: 4 2yuan: 5  
5yuan: 0 5yuan: 1 5yuan: 1 5yuan: 2 5yuan: 2 5yuan: 1 5yuan: 2 5yuan: 2 5yuan: 3 5yuan: 3  
请按任意键继续... . . .
```

输出:

output\_lab4.1.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
3

测试二:

输入:

input\_lab4.1.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
3  
2 5 11  
18

运行结果:

```
c:\VS\code\c and c++\datastructure\lab4.1.exe  
The final result of the dynamic programming calculation is:  
0yuan 1yuan 2yuan 3yuan 4yuan 5yuan 6yuan 7yuan 8yuan 9yuan 10yuan 11yuan 12yuan 13yuan 14yuan 15yuan 16yuan 17yuan 18yuan  
lyuan: 0 lyuan: -1  
2yuan: 0 2yuan: -1 2yuan: 1 2yuan: -1 2yuan: 2 2yuan: -1 2yuan: 3 2yuan: -1 2yuan: 4 2yuan: -1 2yuan: 5 2yuan: -1 2yuan: 6 2yuan: -1  
5yuan: 0 5yuan: -1 5yuan: 1 5yuan: -1 5yuan: 2 5yuan: 1 5yuan: 2 5yuan: 3 5yuan: 2 5yuan: 3 5yuan: 4 5yuan: 3 5yuan: 4 5yuan: 4 5yuan: 5  
18yuan: 0 18yuan: -1 18yuan: 1 18yuan: -1 18yuan: 2 18yuan: -1 18yuan: 3 18yuan: 2 18yuan: 1 18yuan: 2 18yuan: 3 18yuan: 1 18yuan: 2 18yuan: 3 18yuan:  
18yuan: 0 18yuan: -1 18yuan: 1 18yuan: 2 18yuan: 1 18yuan: 3 18yuan: 2 18yuan: 1 18yuan: 2 18yuan: 3 18yuan: 2 18yuan: 1 18yuan: 3 18yuan:
```

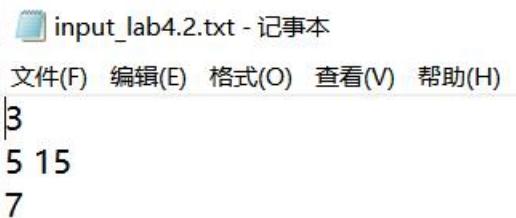
输出:

output\_lab4.1.txt - 记事本  
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)  
3

实验达到预期。

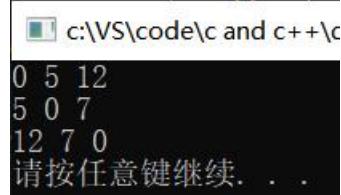
对于实验 4.2，测试结果如下:

输入:



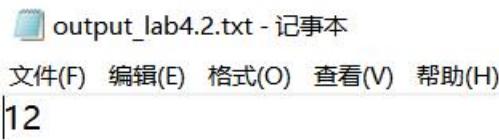
```
input_lab4.2.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
3
5 15
7
```

运行结果:



```
c:\VS\code\c and c++\c
0 5 12
5 0 7
12 7 0
请按任意键继续. . .
```

输出:



```
output_lab4.2.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
12
```

实验达到预期。

(2) 算法实现的复杂度在问题规模很大时可以接受吗?

答:可以接受,因为动态规划算法有很好的效率,所以当问题复杂度很大时,就不会影响到算法的运行时间。

(3) 如果不用动态规划方法还能想到其他的解决方式吗?和动态规划相比会有更好的效率吗?

答:对于找硬币问题,有时候贪心算法也能解决,但不如动态规划求解有效率,所以采用动态规划方法是一个很好的选择。

(4) 所选用的数据结构合适吗?

答:采用了数组的数据结构,合适,因为该数据结构能够支持对于数组中的元素的随机访问,而且方便查询。

(5) 叙述通过实验你对动态规划方法的理解及其优缺点

答:优点:动态规划方法有效利用了子问题的重叠性,减少了大量的计算。而且动态规划的使用也非常简单,只需要问题满足最优子结构、子问题重叠性、无后效性即可。

通常可以利用分治思想构造出子问题的分解方法,就可以利用动态规划;

缺点:动态规划的缺点并不是最快的方法,只是解决某一类型的问题的工具或者优化某些NPC问题的时间效率。动态规划的很重要的一点就是用大量空间换取时间上的优化,所以这并不是一个完美的方法。

## 六、附录

(1) 如果你对这个实验还有其他的解决方案或设想,或对我们的实验方案有什么意见,请在此描述。

在实验4.1中,还编写了求解对于目标钱数和不同面值硬币下,其组合方式数量的函数。具体函数代码如下:

```

int change(vector<int> &money, int target){
    for(int i=0;i<money.size();i++){
        ddpp[i][0]=1;
    }
    for(int j=0;j<=target;j++){
        if(j%money[0]==0){
            ddpp[0][j]=1;
        }else{
            ddpp[0][j]=0;
        }
    }
    for(int i=1;i<money.size();i++){
        for(int j=1;j<=target;j++){
            int temp=0;
            for(int k=0;j-k*money[i]>=0;k++){
                temp+=ddpp[i-1][j-k*money[i]];
            }
            ddpp[i][j]=temp;
        }
    }
    return ddpp[money.size()-1][target];
}

```

此函数也是运用了动态规划的思想方法。

然后对于实现**实验 4.1** 的算法做出了改进：

```

int solution2(vector<int> &money, int target){
    for(int i=1;i<=target;i++){
        res[i]=INT_MAX-10000;
    }
    for(int i=1;i<=target;i++){
        for(int j=0;j<money.size();j++){
            int temp=INT_MAX;
            if(i-money[j]>=0){
                temp=min(res[i-money[j]]+1,res[i]);
                res[i]=temp;
            }
        }
    }
    return res[target];
}

```

显然此算法的时空复杂度均有提升，时间复杂度由  $O(n^3)$  数量级下降为  $O(n^2)$ ，空间复杂度由  $O(n^2)$  数量级下降为  $O(n)$ 。是一个不错的提升，算法效率大大提高。

## (2) 实验参考的资料

### 实验 4.1：

[https://blog.csdn.net/u011426016/article/details/86565062?ops\\_req](https://blog.csdn.net/u011426016/article/details/86565062?ops_req)

uest\_misc=%257B%2522request%255Fid%2522%253A%25221654237078167814  
32984009%2522%252C%2522scm%2522%253A%252220140713.130102334..%252  
2%257D&request\_id=165423707816781432984009&biz\_id=0&utm\_medium=di  
stribute.pc\_search\_result.none-task-blog-2~all~baidu\_landing\_v2~d  
efault-1-86565062-null-null.142^v11^control,157^v13^control&utm\_t  
erm=%E6%89%BE%E9%9B%B6%E9%92%B1%E9%97%AE%E9%A2%98%E5%8A%A8%E6%80%  
81%E8%A7%84%E5%88%92&spm=1018.2226.3001.4187

#### 实验 4.2:

暂无

(3) 实验源代码:

#### 实验 4.1:

```
#include<iostream>
#include<string>
#include<stack>
#include<cstdlib>
#include<sstream>
#include<vector>
#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
#include <limits.h>
#include<fstream>
#include<math.h>
using namespace std;

int dp[1024][1024]={0};
int ddpp[1024][1024]={0};
int test[1024][1024]={0};
int Min(int a,int b){
    return a<=b?a:b;
}
int solution2(vector<int> &money,int target){
    for(int i=1;i<=target;i++){
        res[i]=INT_MAX-10000;
    }
    for(int i=1;i<=target;i++){
        for(int j=0;j<money.size();j++){
            int temp=INT_MAX;
            if(i-money[j]>=0){
                temp=min(res[i-money[j]]+1,res[i]);
                res[i]=temp;
            }
        }
    }
}
```

```

    }

    return res[target];
}

int solution1(vector<int> &money, int target){
    for(int i=0;i<money.size();i++){
        dp[i][0]=0;
    }

    for(int j=0;j<=target;j++){
        if(j%money[0]==0){
            dp[0][j]=j/money[0];
            // dp[0][j]=1;
        }else{
            dp[0][j]=(int)ceil((double)j/money[0]);
        }
    }

    for(int i=1;i<money.size();i++){
        for(int j=1;j<=target;j++){
            int temp=INT_MAX;

            for(int k=0;j-k*money[i]>=0;k++){
                temp=Min(dp[i-1][j-k*money[i]]+k,temp);
            }

            dp[i][j]=temp;
        }
    }

    return dp[money.size()-1][target];
}

```

```

int solution(vector<int> &money, int target){
    for(int i=0;i<money.size();i++){
        dp[i][0]=0;
    }

    for(int j=0;j<=target;j++){
        if(j%money[0]==0){
            dp[0][j]=j/money[0];
            // dp[0][j]=1;
        }else{
            dp[0][j]=(int)ceil((double)j/money[0]);
        }
    }

    for(int i=1;i<money.size();i++){
        for(int j=1;j<=target;j++){
            int temp=INT_MAX;

            for(int k=0;j-k*money[i]>=0;k++){

```

```

        temp=Min(dp[i-1][j-k*money[i]]+k,temp);
        if(j==k*money[i]||test[i-1][j-k*money[i]]==1){
            test[i][j]=1;
        }
    }
    dp[i][j]=temp;
}
}

return dp[money.size()-1][target];
}

void Print_dp(vector<int> &money,int target){
    cout<<"      ";
    for(int i=0;i<=target;i++){
        cout<<i<<"yuan      ";
    }
    cout<<endl;
    for(int j=0;j<money.size();j++){
        for(int i=0;i<=target;i++){
            cout<<money[j]<<"yuan:  ";
            cout<<dp[j][i]<<"  ";
        }
        cout<<endl;
    }
}

void correct(vector<int> &money,int target){
    for(int i=0;i<money.size();i++){
        for(int j=1;j<=target;j++){
            if(test[i][j]==0){
                dp[i][j]=-1;
            }
        }
    }
}

int change(vector<int> &money,int target){
    for(int i=0;i<money.size();i++){
        ddpp[i][0]=1;
    }
    for(int j=0;j<=target;j++){
        if(j%money[0]==0){
            ddpp[0][j]=1;
        }else{
            ddpp[0][j]=0;
        }
    }
}

```

```

        for(int i=1;i<money.size();i++){
            for(int j=1;j<=target;j++){
                int temp=0;
                for(int k=0;j-k*money[i]>=0;k++){
                    temp+=ddpp[i-1][j-k*money[i]];
                }
                ddpp[i][j]=temp;
            }
        }
        return ddpp[money.size()-1][target];
    }

template<class T>
vector<T> &Open_file(string filename,vector<T> &store){
    ifstream in_file;
    in_file.open(filename);
    string item;
    getline(in_file,item);
    int length=stoi(item);
    store.push_back(item);
    for(int i=0;i<length;i++){
        if(i==length-1){
            getline(in_file,item);
        }else{
            getline(in_file,item,' ');
        }
        store.push_back(item);
    }
    getline(in_file,item);
    store.push_back(item);
    return store;
}

void Write_file(string filename,int res){
    ofstream of(filename);
    of<<res<<endl;
    of.close();
}

int main(){
    vector<string> store;
    string filename="L://LAB/input_lab4.1.txt";
    string outfilename="L://LAB/output_lab4.1.txt";
    int money_number;
    vector<int> money;
    int target;
}

```

```

store=Open_file(filename,store);
for(int i=0;i<store.size();i++){
    if(i==0){
        money_number=stoi(store[i]);
    }else if(i==store.size()-1){
        target=stoi(store[i]);
    }else{
        money.push_back(stoi(store[i]));
    }
}
if(money[0]!=1){
    money.insert(money.begin(),1);
}
Write_file(outfilename,solution(money,target));
cout<<"The final result of the dynamic programming calculation is:"<<endl;
correct(money,target);
Print_dp(money,target);
system("pause");
}

```

## 实验 4.2:

```

#include<iostream>
#include<string>
#include<stack>
#include<cstdlib>
#include<sstream>
#include<vector>
#include<stdio.h>
#include<malloc.h>
#include <stdlib.h>
#include <limits.h>
#include<fstream>
#include<math.h>
using namespace std;

int dp[1024][1024];
template<class T>
vector<T> &Open_file(string filename,vector<T> &store){
    ifstream in_file;
    in_file.open(filename);
    string item;
    getline(in_file,item);

```

```

int length=stoi(item);
store.push_back(item);
for(int index=1;index<length;index++){
    for(int i=0;i<length-index;i++){
        if(i==length-index-1){
            getline(in_file,item);
        }else{
            getline(in_file,item,' ');
        }
        store.push_back(item);
    }
}
getline(in_file,item);
store.push_back(item);
return store;
}
void Write_file(string filename,int res){
    ofstream of(filename);
    of<<res<<endl;
    of.close();
}

```

```

int solution(vector<int> &rent,int sum){
    int index=0;
    for(int i=0;i<sum;i++){
        for(int j=i;j<sum;j++){
            if(i==j){
                dp[i][j]=0;
                continue;
            }else{
                dp[i][j]=rent[index];
                index++;
            }
        }
    }
    for(int i=0;i<sum;i++){
        for(int j=i;j<sum;j++){
            if(i==j){
                continue;
            }else{
                int temp=INT_MAX;
                for(int k=i;k<=j;k++){
                    temp=min(temp,dp[i][k]+dp[k][j]);
                }
            }
        }
    }
}

```

```

        dp[i][j]=temp;
    }
}
}

for(int i=0;i<sum;i++){
    for(int j=i;j<sum;j++){
        dp[j][i]=dp[i][j];
    }
}
return dp[0][sum-1];
}

void Print_dp(vector<int> &rent,int sum){
    for(int i=0;i<sum;i++){
        for(int j=0;j<sum;j++){
            cout<<dp[i][j]<<" ";
        }
        cout<<endl;
    }
}

int main(){
    vector<string> store;
    string filename="L://LAB/input_lab4.2.txt";
    string outfilename="L://LAB/output_lab4.2.txt";
    int length;
    vector<int> rent;
    store=Open_file(filename,store);
    for(int i=0;i<store.size();i++){
        if(i==0){
            length=stoi(store[i]);
        }else{
            rent.push_back(stoi(store[i]));
        }
    }
    Write_file(outfilename,solution(rent,length));
    Print_dp(rent,length);
    system("pause");
}

```

教师评语：

评价表格：

评价内容	具体要求	分值	得分
平时表现	课程设计过程中，无缺勤现象，态度积极，具有严谨的学习态度和认真、踏实、一丝不苟的科学作风。	20	
报告质量	实验报告格式规范，符合要求；报告内容充实、正确，实验目的归纳合理到位。	30	
实验内容	能够按实验要求合理设计并开发出程序，功能完整性 强，原理及实验结果分析准确，归纳总结充分。	50	
总 分			