第 2 章　实验内容与指导

上机实验题帮助学生学会如何把课堂上学到的数据结构与算法的基础知识用于解决实际问题，培养软件工作者所需要的实践能力；另一方面，能使课堂上学到的知识变活，达到深化理解和灵活掌握教学内容的目的，并训练学生进行复杂程序设计的技能和培养良好程序设计的风格。

2.1　实验一　线性表的顺序存储

【实验目的】

1．掌握线性表顺序存储结构的特点：逻辑上相邻的数据元素其物理位置上也相邻。
2．掌握线性表顺序存储结构的查找、插入、删除等操作。

【实验内容】

1．输入一组整型元素序列，建立顺序表。
2．遍历该顺序表。
3．在该顺序表中顺序查找某一元素，查找成功返回 1，否则返回 0。
4．实现把该表中所有奇数排在偶数前，即表的前面为奇数，后面为偶数。
5．判断该顺序表中元素是否对称，对称返回 1，否则返回 0。
6．输入整型元素序列，利用有序表插入算法建立一个有序表。
7．利用实验内容 6 创建两个递增有序表，然后将它们合并成一个递增有序表。
8．编写一个主函数，调试上述算法。
9．综合训练：利用顺序表实现一个班级学生信息管理（数据录入、插入、删除、排序、查找等）。

【实验说明】

1．顺序表的类型采用如下结构：

```
#define MAXSIZE 100//表中元素的最大个数
typedef int ElemType;//元素类型
typedef struct list{
    ElemType elem[MAXSIZE];//静态线性表
    int length;//表的实际长度
}SqList;//顺序表的类型名
```

2．建立顺序表时可利用随机函数自动产生数据。
3．实验内容 8、9 是选做内容，可不做要求。

【实验指导】

1．参考程序为：

```
void CreateSqList(SqList *L)
{
    int n,i;
    do{
        printf("请输入数据元素的个数:");
        scanf("%d",&n);
        if(n<=0)printf("输入错误\n");
    }while(n<=0);
    for(i=0;i<n;i++)
        scanf("%d",&(L->elem[i]));
    L->length=n;
}
```

2．参考程序为：

```
void PrintList(SqList L)
{
    int i;
    for(i=0;i<L.length;i++)
        printf("%d",L.elem[i]);
    printf("\n");
}
```

3．参考程序为：

```
int Findelems(SqList L,ElemType e)
{
    int i;
    for(i=0;i<L.length;i++)
        if(L.elem[i]==e)
            return 1;
    return 0;
}
```

4．分析：从顺序表表头开始扫描，当数据元素为偶数时就从该数开始往后查找，一旦找到奇数，则将该偶数与此奇数交换。顺序表中所有数据全部扫描结束后，所有奇数就排列在表的前端。参考程序为：

```
void ChangeVal(SqList *L)
{
    int i,j,temp;
    for(i=0;i<L->length;i++)
    {
        if(L->elem[i]%2==0)
        {
            for(j=i+1;j<L->length;j++)
            {
                if(L->elem[j]%2!=0)
```

```
                    {
                        temp=L->elem[i];
                        L->elem[i]=L->elem[j];
                        L->elem[j]=temp;
                        break;
                    }
                }
                if(j==L->length)break;
            }
        }
    }
```

5．参考程序为：

```
int YesNo_Symmetry(SqList L)
{
    int i,j;
    j=L.length-1;
    for(i=0;i<j/2;i++)
    {
        if(L.elem[i]!=L.elem[j-i])
            return 0;
    }
    return 1;
}
```

6．参考程序为：

```
void Insert_OrderList(SqList *L,int x)
{
    int i,j;
    for(i=0;i<L->length;i++)
        if(L->elem[i]>x)break;
    for(j=L->length-1;j>=i;j--)
        L->elem[j+1]=L->elem[j];
    L->elem[i]=x;
    L->length++;
}
void Create_OrderList(SqList *L)
{
    int n,i,input;
    do{
        printf("请输入数据元素的个数:");
        scanf("%d",&n);
        if(n<=0)printf("输入错误\n");
    }while(n<=0);
    for(i=0;i<n;i++)
    {
        scanf("%d",&input);
        Insert_OrderList(L,input);
```

```
        }
    }
```

7．参考程序为：

```
SqList *Merge_OrderList(SqList A,SqList B)
{//将有序顺序表 A 和 B 合并到有序顺序表 C 中返回
    int i=0,j=0,k=0;
    SqList *C=(SqList *)malloc(sizeof(SqList));
    C->length=0;
    while(j<A.length && k<B.length)
    {
        if(A.elem[j]<B.elem[k])
            C->elem[i++]=A.elem[j++];
        else
            C->elem[i++]=B.elem[k++];
    }
    if(j==A.length)
        while(k<B.length)C->elem[i++]=B.elem[k++];
    if(k==B.length)
        while(j<A.length)C->elem[i++]=A.elem[j++];
    C->length=i;
    return C;
}
```

2.2　实验二　线性表的链式存储

【实验目的】

1．掌握线性链表的操作特点，即指针是逻辑关系的映像。

2．掌握动态产生单链表的方法。

3．熟练掌握单链表的插入、删除操作特点，即指针赋值的先后次序。

【实验内容】

1．分别采用随机函数产生或键盘输入一组整型数，建立一个带头结点的单向链表（无序）。

2．遍历单向链表。

3．把单向链表中元素逆置（不允许申请新的结点空间）。

4．在单向链表中删除所有结点的值为偶数的结点。

5．编写在非递减有序链表中插入一个元素使链表元素仍有序的函数，并利用该函数建立一个非递减有序单向链表。

6．利用实验内容 5 建立两个递增有序单向链表，然后合并成一个递增链表。

7．利用实验内容 1 建立的链表，实现将其分解成两个链表，其中一个全部为奇数，另一个全部为偶数（尽量利用已知的存储空间）。

8．在主函数中设计一个简单的菜单，分别调试上述算法。

【实验说明】

1．链表类型采用如下结构：

```
typedef int ElemType;//元素类型
typedef struct LNode
{
    ElemType data;
    struct LNode *next;
}LNode,*LinkList;
```

2．为了算法实现简单，最好采用带头结点的单向链表。

【实验指导】

1．参考程序为：

```
LinkList CreateListH(void)//头插法产生带头结点单链表
{
    int ch;
    LinkList head=(LinkList)malloc(sizeof(LNode));
    LinkList s;
    head->next=NULL;
    while(scanf("%d",&ch)==1)//输入数据类型错误时结束单链表的生成
    {
        s=(LinkList)malloc(sizeof(LNode));
        s->data=ch;
        s->next=head->next;
        head->next=s;
    }
    return head;
}
LinkList CreateListRand(void)//利用随机函数产生带头结点单链表（头插法）
{
    int ch,i;
    LinkList head=(LinkList)malloc(sizeof(LNode));
    LinkList s;
    head->next=NULL;
    srand((unsigned)time(NULL));
    printf("Please input Create Numbers:");
    scanf("%d",&ch);
    for(i=0;i<ch;i++)
    {
        s=(LinkList)malloc(sizeof(LNode));
        s->data=rand()%50;//随机产生 0～49 之间的数
        s->next=head->next;
        head->next=s;
    }
```

```
    return head;
}
```

2．参考程序为：

```
void PrintLinkList(LNode L)
{
    LinkList p;
    p=L.next;
    while(p)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}
```

3．参考程序为：

```
void Inverse_set(LinkList head)
{
    LNode *r,*m=NULL,*p;
    p=head->next;
    while (p!=NULL)
    {
        r=m; m=p;
        p=p->next;
        m->next=r;
    }
    head->next=m;
}
```

4．参考程序为：

```
void DelEvenLinkList(LinkList head)
{
    LinkList q,p;
    p=head->next;
    q=head;
    while(p)
    {
        if(p->data%2==0)
        {
            q->next=p->next;
            free(p);
            p=q->next;
        }
        else
        {
            q=p;
```

```
                p=p->next;
            }
        }
    }
```

5．参考程序为：

```
void InsertIncr(LinkList head,ElemType x)
{//将结点插入递增的单链表
    LinkList q,p,s;
    s=(LinkList)malloc(sizeof(LNode));
    s->data=x;
    q=head;
    p=head->next;
    while(p && p->data<x)
    {
        q=p;
        p=p->next;
    }
    s->next=q->next;
    q->next=s;
}
LinkList CreateListIncr(void)
{//通过调用插入有序链表函数生成递增单链表
    int ch;
    LinkList head=(LinkList)malloc(sizeof(LNode));
    LinkList s;
    head->next=NULL;
    while(scanf("%d",&ch)==1)//输入数据类型错误时结束单链表的生成
        InsertIncr(head,ch);
    return head;
}
```

6．参考程序为：

```
LinkList LinkListCat(LinkList head1,LinkList head2)
{
    LinkList h1,h2,h;
    LinkList head=(LinkList)malloc(sizeof(LNode));
    head->next=NULL;
    h1=head1->next;
    h2=head2->next;
    h=head;
    while(h1 && h2)
    {
        if(h1->data<h2->data)
        {
            h->next=h1;
```

```
            h=h->next;
            h1=h1->next;
        }
        else
        {
            h->next=h2;
            h=h->next;
            h2=h2->next;
        }
    }
    if(h1)h->next=h1;
    if(h2)h->next=h2;
    return head;
}
```

7．参考程序为：

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
typedef int ElemType;//元素类型
typedef struct LNode
{
    ElemType data;
    struct LNode *next;
}LNode,*LinkList;
void PrintLinkList(LNode L)
{
    LinkList p;
    p=L.next;
    while(p)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}
void DecoLinkList(LNode head,LinkList head1,LinkList head2)
{//将单链表 head 拆分奇数链 head1 和偶数链 head2
    LinkList h,h1,h2;
    h=head.next;
    h1=head1;
    h2=head2;
    while(h)
    {
        if(h->data%2==0)
```

```
            {
                h2->next=h;
                h=h->next;
                h2=h2->next;
            }
            else
            {
                h1->next=h;
                h=h->next;
                h1=h1->next;
            }
        }
        h1->next=NULL;
        h2->next=NULL;
}
main()
{
        LinkList head;
        LinkList head1=(LinkList)malloc(sizeof(LNode));
        LinkList head2=(LinkList)malloc(sizeof(LNode));
        head=CreateListIncr();
        PrintLinkList(*head);
        DecoLinkList(*head,head1,head2);
        PrintLinkList(*head1);
        PrintLinkList(*head2);
}
```

2.3 实验三 栈

【实验目的】

1．掌握栈这种数据结构的特性及其主要存储结构。

2．掌握顺序栈、链栈的初始化、入栈和出栈操作，并能在现实生活中灵活运用。

【实验内容】

1．采用顺序存储实现栈的初始化、入栈、出栈操作。

2．采用链式存储实现栈的初始化、入栈、出栈操作。

3．写一个程序，将输入的十进制数据 M 转换为八进制数据 M8，将其调试通过。在此基础上修改程序，实现十进制数据 M 向 N 进制（二、八或十六进制）的转换。

（1）采用顺序存储结构实现栈。

（2）采用链表结构实现栈。

4．将教材算法 3-19 转换得到的后缀表达式，利用顺序栈求表达式的值。

【实验说明】

1．顺序栈类型采用如下结构：

```
#define Stack_Size 100
typedef struct Stack
{
    ElemType    elem[Stack_Size];     //用来存放栈中元素的一维数组
    int    top;                       //用来存放栈顶元素的下标
}SqStack;
```

2．链栈类型采用如下结构：

```
typedef struct StackNode
{
    ElemType data;
    struct StackNode *next;
}StackNode;
typedef struct
{
    StackNode *top;    //栈顶指针
}LinkStack;
```

【实验指导】

1．参考程序为：

```
#include<stdio.h>
#include<stdlib.h>
#define Stack_Size 100
#define OK       1
#define ERROR       0
typedef int ElemType;
typedef struct Stack
{
ElemType elem[Stack_Size];        //用来存放栈中元素的一维数组
int    top;                       //用来存放栈顶元素的下标
}SqStack;
int InitStack(SqStack **s)//初始化顺序栈
{
    (*s)=(SqStack *)malloc(sizeof(SqStack));
     if((*s)==NULL)return ERROR;
    (*s)->top=-1;
    return OK;
}
int EmptyStack(SqStack s)//判断栈空
{
    if(s.top==-1)
    {
        printf("stack is empty!\n");
```

```
            return OK;
        }
        return ERROR;
}
int GetTop(SqStack s,int *e)//取栈顶元素
{
    if(EmptyStack(s))return ERROR;
    *e=s.elem[s.top];
    return OK;
}
int Push(SqStack *s,int e)//入栈
{
    if(s->top==Stack_Size-1)
    {
        printf("stack is full!\n");
        return ERROR;
    }
    s->top++;
    s->elem[s->top]=e;
    return OK;
}
void PrintStack(SqStack s)//打印栈中数据
{
    int i;
    for(i=0;i<=s.top;i++)
        printf("%d ",s.elem[i]);
    printf("\n");
}
int Pop_Stack(SqStack *s,int *e)//出栈
{
    if(EmptyStack(*s))
        return ERROR;
    *e=s->elem[s->top];
    s->top--;
    return OK;
}
void main()
{
    int cord,e,x,y;
    SqStack *s;
    do
    {
        printf("\n Main Menu    \n");
        printf("1----Creat Stack\n");
        printf("2----Get Top Element\n");
        printf("3----Push\n");
```

```
            printf("4----Pop\n");
            printf("5----Print\n");
            printf("6----quit\n");
            scanf("%d",&cord);
            switch(cord)
            {
                case 1:
                        InitStack(&s);
                        break;
                case 2:
                        if(GetTop(*s,&y))
                           printf("Stack Top=[%d]\n",y);
                        break;
                case 3:
                        printf("Please input push element:");
                        scanf("%d",&e);
                        Push(s,e);
                        break;
                case 4:
                        if(Pop_Stack(s,&x))
                           printf("Pop stack=[%d]\n",x);
                        break;
                case 5:
                        PrintStack(*s);
                        break;
                case 6:
                        return;
            }
        }while(cord<=6);
    }
```

2．参考程序为：

```
    #include<stdio.h>
    #include<stdlib.h>
    #define Stack_Size 100
    #define OK       1
    #define ERROR       0
    typedef int ElemType;
    typedef struct stacknode
    {
        ElemType data;
        struct stacknode *next;
    }StackNode;
    typedef struct
    {
        StackNode *top;/*栈顶指针*/
    }LinkStack;
```

```
void InitStack(LinkStack *s)//初始化栈
{
    s->top=NULL;
}
int EmptyStack(LinkStack s)//判断栈空
{
    if(s.top==NULL)return OK;
    else return ERROR;
}
int GetTop(LinkStack s,int *e)//取栈顶元素
{
    if(EmptyStack(s))return ERROR;
    *e=s.top->data;
}
void Push(LinkStack *s,int e)//入栈
{
    StackNode *p=(StackNode *)malloc(sizeof(StackNode));
    p->data=e;
    p->next=s->top;
    s->top=p;
}
int Pop_Stack(LinkStack *s,int *e)//出栈
{
    StackNode *p;
    if(EmptyStack(*s))return ERROR;
    p=s->top;
    *e=p->data;
    s->top=p->next;
    free(p);
    return OK;
}
void PrintStack(LinkStack s)//打印栈中元素
{
    StackNode *p=s.top;
    while(p)
    {
        printf("%d ",p->data);
        p=p->next;
    }
}
void main()
{
    int cord,e,x,y;
    LinkStack s;
    do
```

```
{
    printf("\n Main Menu    \n");
    printf("1----Creat Stack\n");
    printf("2----Get Top Element\n");
    printf("3----Push\n");
    printf("4----Pop\n");
    printf("5----Print\n");
    printf("6----quit\n");
    scanf("%d",&cord);
    switch(cord)
    {
        case 1:
            InitStack(&s);
            break;
        case 2:
            if(GetTop(s,&y))
              printf("Stack Top=[%d]\n",y);
            break;
        case 3:
            printf("Please input push element:");
            scanf("%d",&e);
            Push(&s,e);
            break;
        case 4:
            if(Pop_Stack(&s,&x))
              printf("Pop stack=[%d]\n",x);
            break;
        case 5:
            PrintStack(s);
            break;
        case 6:
            return;
    }
}while(cord<=6);
}
```

3．参考程序为：

（1）

```
void Conversion(SqStack *S)
{
    int N,n1,t;
    printf("输入一个十进制数字:\n");
    scanf("%d",&N);//输入一个十进制数字
    printf("输入要转换的 n 进制数字(2、8、16):\n");
    scanf("%d",&n1);//输入要转换的进制
    while(N)
```

```
        {
            Push(S,N%n1);
            N=N/n1;
        }
        printf("该数转化为%d 进制数为:\t",n1);
        if(n1==16)
        {
            while(!EmptyStack(*S))
            {
                Pop_Stack(S,&t);
                if(t==10){printf("A ");continue;}
                if(t==11){printf("B ");continue;}
                if(t==12){printf("C ");continue;}
                if(t==13){printf("D ");continue;}
                if(t==14){printf("E ");continue;}
                if(t==15){printf("F ");continue;}
                printf("%d ",t);
            }
        }
        else
            PrintStack(*S);
    }
    void main()
    {
        SqStack *S;
        InitStack(&S);
        Conversion(S);
    }
```

（2）

```
    void Conversion(LinkStack *S)
    {
        int N,n1,t;
        printf("输入一个十进制数字:\n");
        scanf("%d",&N);//输入一个十进制数字
        printf("输入要转换的 n 进制数字(2、8、16):\n");
        scanf("%d",&n1);//输入要转换的进制
        while(N)
        {
            Push(S,N%n1);
            N=N/n1;
        }
        printf("该数转化为%d 进制数为:\t",n1);
        if(n1==16)
        {
            while(!EmptyStack(*S))
            {
```

```
                Pop_Stack(S,&t);
                if(t==10){printf("A "); continue;}
                if(t==11){printf("B "); continue;}
                if(t==12){printf("C "); continue;}
                if(t==13){printf("D "); continue;}
                if(t==14){printf("E "); continue;}
                if(t==15){printf("F "); continue;}
                printf("%d ",t);
            }
        }
        else
            PrintStack(*S);
    }
    void main()
    {
        LinkStack S;
        InitStack(&S);
        Conversion(&S);
    }
```

4．参考程序为：

```
#include<stdio.h>
#include<stdlib.h>
#define True 1
#define False -1
#define Stack_Size 100
#define OK      1
#define ERROR       0
typedef float ElemType;
typedef struct Stack
{
    ElemType elem[Stack_Size];          //用来存放栈中元素的一维数组
    int    top;                         //用来存放栈顶元素的下标
}SqStack;
int InitStack(SqStack **s)
{
    if((*s=(SqStack *)malloc(sizeof(SqStack)))==NULL)
        return ERROR;
    (*s)->top=-1;
    return OK;
}
int EmptyStack(SqStack s)
{
    if(s.top==-1)
        return OK;
     return ERROR;
}
```

```
int GetTop(SqStack s,ElemType *e)
{
    if(EmptyStack(s))return ERROR;
    *e=s.elem[s.top];
    return OK;
}
int Push(SqStack *s,ElemType e)
{
    if(s->top==Stack_Size-1)
        return ERROR;
    s->top++;
    s->elem[s->top]=e;
    return OK;
}
int Pop(SqStack *s,ElemType *e)
{
    if(EmptyStack(*s))
        return ERROR;
    *e=s->elem[s->top];
    s->top--;
    return OK;
}
float Operate(float a,char theta,float b)
{
    switch(theta)
    {
        case '+': return a+b;
        case '-': return a-b;
        case '*': return a*b;
        case '/': return a/b;
        default: return ERROR;
    }
}
int In(char c)
{
    switch(c)
    {
        case '#':
        case ')':
        case '+':
        case '-':
        case '*':
        case '/':
        case '(': return True;
        case ' ': return False;
        default : return ERROR;
```

```
        }
    }
    float Evaluateexpression_r(char *S)
    {
        SqStack *OPND;
        float a,b; int k; char *p=S,c;
        InitStack(&OPND);
        c=*p;
        while(c!='#')
        {
            if(!In(c))
            {
                for(k=0;In(c)!=-1;c=*++p)
                    k=10*k+c-'0';
                Push(OPND,k);
            }
            else
            {
                Pop(OPND,&b);Pop(OPND,&a);
                Push(OPND,Operate(a,c,b));
            }
            c=*++p;
        }
        GetTop(*OPND,&a);
        return a;
    }
    void main()
    {
        char S1[100]={0};
        float result;
        puts("Please input suffix expression:");
        gets(S1);
        result=Evaluateexpression_r(S1);
        printf("%.2f\n",result);
    }
```

2.4　实验四　队列

【实验目的】

1．掌握队列这种数据结构的逻辑特性及其主要存储结构。
2．在简单情况下会使用顺序结构实现队列，熟练掌握循环队列的使用。
3．在复杂情况下会使用链表结构的队列，并能在现实生活中灵活运用。

【实验内容】

1．采用顺序存储实现循环队列的初始化、入队、出队操作。

2．采用链式存储实现队列的初始化、入队、出队操作。

3．编写一个程序，使用两个链队 q1 和 q2，用来分别存储由计算机随机产生的 20 个 100 以内的奇数和偶数，然后每行输出 q1 和 q2 的一个值，即奇数和偶数配对输出，直到任一队列为空为止。

【实验说明】

1．循环队列类型采用如下结构：

```
#define MAXSIZE 100//最大队列长度
typedef int ElemType;
typedef   struct{
        ElemType   data[MAXSIZE];
        int   front,rear;//队头、队尾指针
}SqQueue;
```

2．链队类型采用如下结构：

```
typedef struct QNode
{
        ElemType data;
        struct QNode *next;
}QNode,*QueuePtr;
typedef struct
{
        QueuePtr front;
        QueuePtr rear;
}LinkQueue;
```

【实验指导】

1．参考程序为：

```
int InitQueue (SqQueue **Q)//初始化循环队列
{
        *Q=(SqQueue *)malloc(sizeof(SqQueue));
        if(!(*Q))
                return 0;
        (*Q)->front = (*Q)->rear=0;
        return 1;
}
int QueueEmpty(SqQueue Q)//判断队空
{
        return Q.front==Q.rear;
}
int QueueFull(SqQueue Q)//判断队满
{
```

```
    return (Q.rear+1)%MAXSIZE==Q.front;
}
int EnQueue(SqQueue *Q, ElemType e)//入队操作
{
    if (QueueFull(*Q))
        return 0;//队列满
    Q->data[Q->rear]=e;
    Q->rear=(Q->rear+1)%MAXSIZE;
    return 1;
}
int DeQueue (SqQueue *Q, ElemType *e)//出队操作
{
    if (QueueEmpty(*Q))
        return 0;
    else
    {
        *e=Q->data[Q->front];
        Q->front=(Q->front+1)%MAXSIZE;
        return 1;
    }
}
```

2．参考程序为：

```
int InitQueue(LinkQueue *Q)//将 Q 初始化为一个空的链队列
{
    Q->front=Q->rear=(QueuePtr)malloc(sizeof(QNode));
    if (Q->front==NULL)
        return 0;
    Q->front->next=NULL;
    return 1;
}
int QueueEmpty(LinkQueue Q)//判断队空
{
    return Q.front==Q.rear;
}
int EnQueue(LinkQueue *Q, ElemType e)//入队操作
{
    QueuePtr p;
    p=(QueuePtr)malloc(sizeof(QNode));
    if(!p)
        return 0;
    p->data=e;
    p->next=NULL;
    Q->rear->next=p;
    Q->rear =p;
    return 1;
}
```

```
int DeQueue(LinkQueue *Q,ElemType *e)//出队操作
{
    QueuePtr p;
    if(QueueEmpty(*Q))
        return 0;//若队列 Q 为空队列
    p=Q->front->next;
    *e=p->data;
    Q->front->next=p->next;
    if(Q->rear==p)
        Q->rear=Q->front;//若 Q 只有一个结点
    free(p);
    return 1;
}
```

3．参考程序为：

```
int main()
{
    LinkQueue q1,q2;
    int i=0,j=0,num;
    InitQueue(&q1);
    InitQueue(&q2);
    srand((unsigned)time(NULL));
    while( i<20 || j<20 )
    {
        num=rand()%100;
        if(num%2==0&&i<20)
        {
            EnQueue(&q1,num);
            i++;
        }
        if(num%2!=0&&j<20)
        {
            EnQueue(&q2,num);
            j++;
        }
    }
    while(!QueueEmpty(q1)&&!QueueEmpty(q2))
    {
        DeQueue(&q1,&i);DeQueue(&q2,&j);
        printf("%3d %3d\n",i,j);
    }
    free(q1.front);
    free(q2.front);
    return 0;
}
```

2.5　实验五　串与数组

【实验目的】

1．熟悉 C 语言的字符和字符串处理的原理和方法。

2．掌握串的特点及顺序定长存储的方式。

3．熟悉稀疏矩阵的“三元组表”和“十字链表”存储结构，运用它们进行矩阵简单运算处理，如实现矩阵的转置、两个矩阵的相加。

【实验说明】

1．串的定长顺序存储结构如下：

```
#define MAXLEN 256
typedef struct{//串结构定义
char ch[MAXLEN];
int len;
}SString;
```

2．稀疏矩阵三元组表存储结构如下：

```
#define MAXSIZE 12500     //非零元素的最多个数
typedef struct{
        int i,j;                    //该非零元素的行下标和列下标
        ElemType e;                 //该非零元素的值
}Triple;
typedef struct{
        Triple data[MAXSIZE+1];   //非零元素的三元组表。data[0]未用
        int mu,nu,tu;                    //矩阵的行数、列数和非零元素的个数
}TSMatrix;
```

稀疏矩阵十字链表存储结构如下：

```
typedef struct OLNode{
        int i,j;            //非零元素的行和列下标
        ElemType e;
        struct OLNode *right,*down;//非零元素所在行表、列表的后继链域
}OLNode,*OLink;
typedef struct{
        OLink *rhead,*chead;          //行、列链表的头指针向量
        int mu,nu,tu;                    //稀疏矩阵的行数、列数、非零元素的个数
}CrossList;
```

【实验内容】

1．下面是有关串的程序，请进行阅读并进行填空，再上机调试：

```
#define maxsize 50
#include "stdio.h"
int StrLength(char s[])
```

```
{     int i;
      for(i=0;s[i]!='\0';i++);
          return(i);
}
int StrConcat1(char *s1,char *s2,char *s)
{       int i=0,j,len1,len2;
         len1=StrLength(s1);
         len2=StrLength(s2);
         if(_____________)
               return(0);
        j=0;
        while(s1[j]!='\0')
        {_____________;    i++;    j++;}
         _____________
        while(s2[j]!='\0')
        {s[i]=s2[j];    i++;     j++;}
        s[i]='\0';
        return 1;
}
int StrSub (char *t, char *s, int i, int len)
/*用 t 返回串 s 中第个 i 字符开始的长度为 len 的子串 1≤i≤串长*/
{     int slen;int j;
      slen=StrLength(s);
      if (_____________)
      {printf("参数不对"); return 0;}
      for (j=0; j<len; j++)
            _____________;
      t[j]='\0';
      return 1;}
int    StrIndex_BF(char *s,char *t,int pos)
/*从串 s 的第 pos 个字符开始找首次与串 t 相等的子串*/
{     int i=pos,j=0;
      while (s[i+j]!='\0' && t[j]!='\0')/*都没遇到结束符*/
      {       if (s[i+j]==t[j]) j++;/*继续*/
               else       { i++;j=0; };/*回溯*/
      }
      if (t[j]=='\0')    return (i);/*匹配成功，返回存储位置*/
      else    return -1;
}
void prin(char *s)
{     int i;
      printf("\n");
      for(i=0;s[i]!='\0';i++)
          printf(" %c",s[i]);
}
void main()
```

```
{
    char s1[50]="THIS IS",s2[50]=" A BOOK",s[50],t[50];
    StrConcat1(s1,s2,s);
    prin(s1);prin(s2);prin(s);
    StrSub(t,s,6,2);
    prin(t);
    printf("\n %d",StrIndex_BF(s,t,1));
    printf("\n %d",StrIndex_BF(s,t,5));
}
```

2．要求使用串的定长顺序存储完成串的插入、串删除、串复制、串比较及求子串函数，并编写主函数上机调试。

3．稀疏矩阵采用三元组表存储，编写算法完成两个矩阵加法运算。

4．稀疏矩阵采用十字链表存储，编写算法完成矩阵的转置。

【实验指导】

1．参考代码为：

```
len1==0&&len2==0
s[i]=s1[j]
j=0;
i>slen || i<len
t[j]=s[j]
```

2．参考程序为：

```
int StrInsert(SString *s,int pos,SString t)//在串 s 中序号为 pos 的字符之前插入串 t
{
    int i;
    if(pos<0 || pos>s->len)
        return(0);//插入位置不合法
    if(s->len+t.len<=MAXLEN)
    {//插入后串长≤MAXLEN
        for(i=s->len+t.len-1;i>=t.len+pos;i--)
            s->ch[i]=s->ch[i-t.len];
        for(i=0;i<t.len;i++)
            s->ch[i+pos]=t.ch[i];
        s->len=s->len+t.len;
    }
    else if(pos+t.len<=MAXLEN)
    {//插入后串长>MAXLEN，但串 t 的字符序列可以全部插入
        for(i=MAXLEN-1;i>t.len+pos-1;i--)
            s->ch[i]=s->ch[i-t.len];
        for(i=0;i<t.len;i++)
            s->ch[i+pos]=t.ch[i];
        s->len=MAXLEN;
    }
    else
    {//串 t 的部分字符序列要舍弃
```

```
            for(i=0;i<MAXLEN-pos;i++)
                s->ch[i+pos]=t.ch[i];
            s->len=MAXLEN;
        }
        return(1);
}
int StrDelete(SString *s,int pos,int len)//在串 s 中删除从序号 pos 起 len 个字符
{
        int i;
        if(pos<0 || pos>(s->len-len))
            return(0);
        for(i=pos+len;i<s->len;i++)
            s->ch[i-len]=s->ch[i];
        s->len=s->len-len;
        return(1);
}
void StrCopy(SString *s,SString t)//将串 t 的值复制到串 s 中
{
        int i;
        for (i=0;i<t.len;i++)
            s->ch[i]=t.ch[i];
        s->len=t.len;
}
int StrCompare(SString s,SString t)//若串 s 和 t 相等返回 0，若 s>t 返回 1，若 s<t 返回-1
{
        int i;
        for(i=0;i<s.len&&i<t.len;i++)
            if(s.ch[i]!=t.ch[i])
                return(s.ch[i]-t.ch[i]);
        return(s.len - t.len);
}
int SubString(SString *sub,SString s,int pos,int len)
{//将串 s 中序号 pos 起 len 个字符复制到 sub 中
        int i;
        if(pos<0 || pos>s.len || len<1 || len>s.len-pos)
        {
            sub->len=0;
            return(0);
        }
        else
        {
            for(i=0;i<len;i++)
                sub->ch[i]=s.ch[i+pos];
            sub->len=len;
            return(1);
        }
}
```

3．参考程序为：

```
void Mat_Add(TSMatrix *c,TSMatrix a,TSMatrix b)
{//采用三元组表存储的矩阵 a 和 b 相加，结果放入矩阵 c 中
    int m=1,n=1;
    c->tu=0;
    while(m<=a.tu || n<=b.tu)
    {
        printf("%d,%d %d,%d\n",a.data[m].i,a.data[m].j,b.data[n].i,b.data[n].j);
        if(a.data[m].i==b.data[n].i)
        {
            if(a.data[m].j==b.data[n].j)
            {
                if(a.data[m].e+b.data[n].e!=0)
                {
                    c->tu++;
                    c->data[c->tu].i=a.data[m].i;
                    c->data[c->tu].j=a.data[m].j;
                    c->data[c->tu].e=a.data[m].e+b.data[n].e;
                    m++;n++;
                }
            }
            else if(a.data[m].j<b.data[n].j)
            {
                c->tu++;
                c->data[c->tu].i=a.data[m].i;
                c->data[c->tu].j=a.data[m].j;
                c->data[c->tu].e=a.data[m].e;
                m++;
            }
            else
            {
                c->tu++;
                c->data[c->tu].i=b.data[n].i;
                c->data[c->tu].j=b.data[n].j;
                c->data[c->tu].e=b.data[n].e;
                n++;
            }
            continue;
        }
        if( (a.data[m].i!=0 && a.data[m].i<b.data[n].i) || b.data[n].i==0)
        {
            c->tu++;
            c->data[c->tu].i=a.data[m].i;
```

```
                c->data[c->tu].j=a.data[m].j;
                c->data[c->tu].e=a.data[m].e;
                m++;
                continue;
            }
            if( (b.data[n].i!=0 && a.data[m].i>b.data[n].i) || a.data[m].i==0)
            {
                c->tu++;
                c->data[c->tu].i=b.data[n].i;
                c->data[c->tu].j=b.data[n].j;
                c->data[c->tu].e=b.data[n].e;
                n++;
            }
        }
    }
```

4．参考程序为：

```
int TransposeSMatrix(CrossList *T)
{
    int u,i;
    OLink *head,p,q,r;
    u=T->mu; //交换(*T).mu 和(*T).nu
    T->mu=T->nu;
    T->nu=u;
    head=T->rhead; //交换(*T).rhead 和(*T).chead
    T->rhead=T->chead;
    T->chead=head;
    for(u=1;u<=T->mu;u++) //对 T 的每一行
    {
        p=T->rhead[u]; //p 为行表头
        while(p) //没到表尾，对 T 的每一结点
        {
            q=p->down; //q 指向下一个结点
            i=p->i; //交换.i 和.j
            p->i=p->j;
            p->j=i;
            r=p->down; //交换.down 和.right
            p->down=p->right;
            p->right=r;
            p=q; //p 指向下一个结点
        }
    }
    return 1;
}
```

2.6　实验六　二叉树

【实验目的】

1．掌握二叉树的存储实现。
2．掌握二叉树的遍历思想。
3．掌握二叉树的常见算法的程序实现。

【实验说明】

1．二叉树采用二叉链表存储结构如下：

```
typedef char ElemType;
typedef struct node
{
    ElemType data;
    struct node *lchild;
    struct node *rchild;
}BTNode,*BTree;
```

2．线索二叉树的存储结构如下：

```
typedef char ElemType;
typedef struct BiThrNode{
    int LTag;
    int RTag;
    ElemType data;
    struct BiThrNode *lchild;
    struct BiThrNode *rchild;
}BiThrNode,*BiThrTree;
```

【实验内容】

1．数据域为字符的一棵二叉树用广义表形式输入，创建一个采用二叉链表存储的二叉树，并按广义表的形式输出这棵二叉树。

2．在实验内容 1 的基础上完成这棵二叉树的中序遍历的递归算法。

3．在实验内容 1 的基础上完成这棵二叉树的中序遍历的非递归算法。

4．求二叉树的宽度。

5．求任意二叉树中第一条最长的路径长度，并输出此路径上各结点的值。

6．输出二叉树中从每个叶子结点到根结点的路径。

7．建立前序线索二叉树，并实现前序遍历。

【实验指导】

1．参考代码为：

```
#define MaxSize 100
```

```
void CreateBTNode(BTree *b,char *str)
{//广义表形式输入二叉树，按二叉链表存储二叉树
    BTNode *St[MaxSize],*p=NULL;
    int top=-1,k,j=0;
    char ch;
    *b=NULL;
    ch=str[j];
    while(ch!='\0')
    {
        switch(ch)
        {
        case'(':top++;St[top]=p;k=1;break;
        case')':top--;break;
        case',':k=2;break;
        default:p=(BTNode *)malloc(sizeof(BTNode));
            p->data=ch;p->lchild=p->rchild=NULL;
            if(*b==NULL)
                *b=p;
            else
            {
                switch(k)
                {
                case 1:St[top]->lchild=p;break;
                case 2:St[top]->rchild=p;break;
                }
            }
        }
        j++;
        ch=str[j];
    }
}
void DispBTNode(BTNode *b)//广义表输出二叉树
{
    if(b!=NULL)
    {
        printf("%c",b->data);
        if(b->lchild!=NULL||b->rchild!=NULL)
        {
            printf("(");
            DispBTNode(b->lchild);
            if(b->rchild!=NULL) printf(",");
            DispBTNode(b->rchild);
            printf(")");
        }
    }
}
```

2．参考代码为：

```
void InOrder(BTree T)//中序递归遍历
{
    if (T)
    {
        InOrder(T->lchild);/*中序遍历左子树*/
        printf("%3c",T->data);/*访问根结束*/
        InOrder(T->rchild);/*中序遍历右子树*/
    }
}
```

3．参考代码为：

```
void InOrder1(BTree T)//非递归中序遍历
{
    SqStack *S;BTree P=T;
    InitStack(&S);
    do{
        while(P){   /*从树或子树根出发往左到叶子*/
            Push(S,P);
            P=P->lchild;
        }
        if(S->top!=-1){   /*P 为 NULL 要么是叶子，要么是没有左子树*/
            Pop(S,&P);
            printf("%3c",P->data);
            P=P->rchild;
        }
    }while((S->top!=-1)|| P);
}
```

4．参考代码为：

```
int BTWidth(BTNode *b)   //求二叉树宽度
{
    struct
    {
        int lno;   //结点的层次编号
        BTNode *p;   //结点指针
    }Qu[MaxSize];   //定义顺序非循环队列
    int front,rear;   //定义队首和队尾指针
    int lnum,max,i,n;
    front=rear=0;   //置队列为空
    if(b!=NULL)
    {
        rear++;
        Qu[rear].p=b;   //根结点指针入队
        Qu[rear].lno=1;   //根结点的层次编号为 1
        while(rear!=front)   //队列不为空
        {
            front++;
```

```
            b=Qu[front].p;   //队头出队
            lnum=Qu[front].lno;
            if(b->lchild!=NULL)   //左孩子入队
            {
                rear++;
                Qu[rear].p=b->lchild;
                Qu[rear].lno=lnum+1;
            }
            if(b->rchild!=NULL)   //右孩子入队
            {
                rear++;
                Qu[rear].p=b->rchild;
                Qu[rear].lno=lnum+1;
            }
        }
        max=0;lnum=1;i=1;
        while(i<=rear)
        {
            n=0;
            while(i<=rear&&Qu[i].lno==lnum)
            {
                n++;i++;   //求每层的结点数
            }
            lnum=Qu[i].lno;
            if(n>max) max=n;
        }
        return max;
    }
    else
        return 0;
}
```

5．参考代码为：

```
int BTNodeDepth(BTNode *b)//求二叉树 b 的深度
{
    int lchilddep,rchilddep;
    if(b==NULL)
        return(0);
    else
    {
        lchilddep=BTNodeDepth(b->lchild); //左子数的高度
        rchilddep=BTNodeDepth(b->rchild); //右子树的高度
        return(lchilddep>rchilddep)?(lchilddep+1):(rchilddep+1);
    }
}
void Long(BTree T)
{
```

```
    if(T!=NULL)//在 T 不为空的情况下
    {
        printf("%3c",T->data);//访问结点
        if(BTNodeDepth(T->lchild)>BTNodeDepth(T->rchild))//判断往左走还是往右走
            Long(T->lchild);
        else
            Long(T->rchild);
    }
}
```

6．参考代码为：

```
void PrintStack(SqStack *S)//使用线性栈辅助操作
{
    int i;
    for(i=0;i<=S->top;i++)
        printf("%3c",S->elem[i]);
    printf("\n");
}
void AllPath(BTree T, SqStack *S)//输出二叉树上从根到所有叶子结点的路径
{
  char ch;
  if(T)
  {
    Push(S,T->data);
    if(!T->lchild&&!T->rchild)//如果左指针和右指针同时为空，才说明该结点为叶子结点
        PrintStack(S);
    else
    {
      AllPath(T->lchild,S);
      AllPath(T->rchild,S);
    }
    Pop(S,&ch);
  }
}
```

7．参考代码为：

```
BiThrTree pre;
void PreThreading(BiThrTree p)//先序线索化
{
    if(p)
    {
        if(!p->lchild)
        {   p->LTag=Thread;
            p->lchild=pre;
        }//前驱线索
        if(!pre->rchild)
        {   pre->RTag=Thread;
```

```
                pre->rchild=p;
            }//后继线索
            pre=p;
            if(p->LTag==Link)
                PreThreading(p->lchild);//左子树线索化
            if(p->RTag==Link)
                PreThreading(p->rchild);//右子树线索化
        }
}
BiThrTree PreOrderThreading(BiThrTree T)//先序线索二叉树
{
        BiThrTree thrt;
        if(!(thrt=(BiThrTree)malloc(sizeof(BiThrNode))))
                return NULL;
        thrt->LTag=Link;
        thrt->RTag=Thread;//建头结点
        thrt->rchild=thrt;//右指针回指
        if(!T)
                thrt->lchild=thrt;//空二叉树
        else
        {       thrt->lchild=T;
                pre=thrt;
                PreThreading(T);//先序遍历进行先序线索化
                pre->rchild=thrt;pre->RTag=Thread;//最后一个结点线索化
                thrt->rchild=pre;
        }
        return thrt;
}
void PreOrderTraverse_Thr(BiThrTree thrt)//先序遍历二叉树
{
        BiThrTree p;
        printf("先序遍历结果为：  ");
        p=thrt->lchild;
       while(p!=thrt)
        {
                printf("%3c",p->data);
                while(p->LTag==Link)
                {
                        p=p->lchild;
                        printf("%3c ",p->data);
                }
                p=p->rchild;
        }
        printf("\n");
}
```