**线性表的存储结构定义及基本操作**

**一、实验目的：**

1. 掌握线性表的逻辑特征
2. 掌握线性表顺序存储结构的特点,熟练掌握顺序表的基本运算
3. 熟练掌握线性表的链式存储结构定义及基本操作
4. 理解循环链表和双链表的特点和基本运算
5. 加深对顺序存储数据结构的理解和链式存储数据结构的理解, 逐步培养解决实际问题的编程能力

**二、实验内容**

**(一) 基本实验内容(顺序表)：**

建立顺序表,完成顺序表的基本操作：初始化、**插入、删除**、逆转、输出、销毁、置空表、求表长、查找元素、判线性表是否为空、实现顺序表元素的逆转。

插入操作分成两种：根据顺序表结点的位置插入一个新结点(位置插入)，也可以根据给定的值进行插入( 值插入)；

删除操作也分成两种：根据顺序表结点的位置删除一个结点(位置删除)，也可以根据给定的值删除对应的第一个结点，或者删除指定值的所有结点(值删除)。

**(二) 基本实验内容(单链表)：**

建立单链表,完成链表(带表头结点) 的基本操作：**建立链表**（一个一个地输入各结点数据，并建立起前后相互链接的关系）、**插入、删除**、查找、输出、求前驱、求后继、两个有序链表的合并操作。

其他基本操作还有销毁链表、将链表置为空表、求链表的长度、获取某位置结点的内容、搜索结点。

插入操作分成两种：根据顺序表结点的位置插入一个新结点(位置插入)，也可以根据给定的值进行插入( 值插入)；

删除操作也分成两种：根据顺序表结点的位置删除一个结点(位置删除)，也可以根据给定的值删除对应的第一个结点，或者删除指定值的所有结点(值删除)。

**(三) 扩展实验内容(顺序表)**

根据给定元素的值查前驱元素、查后继元素；顺序表合并等。

关于合并操作：对已建好的两个顺序表进行合并操作,若原线性表中元素非递减有序排列,要求合并后的结果还是有序(有序合并); 对于原顺序表中元素无序排列的合并只是完成 A=A ∪B ( 无序合并), 要求同样的数据元素只出现一次。

**(四) 扩展实验内容(单链表)**

根据给定结点的值查前驱元素、查后继元素；两个有序链表的合并（分别将两个单链表的结点依次插入到第 3 个单链表中,继续保持结点有序）。

**三、实验环境和实验步骤**

编程语言：C、C++、Java、python等均可以

以上每一项实验内容编写一个主程序，该项试验内容所涉及的每个操作编写一个函数，主程序调用各不同的算法。实验数据自定。

比如，主程序界面：

1. 创建空表；
2. 追加新元素；
3. 插入新元素；
4. 删除元素；
5. 查找元素；
6. 清空表；
7. 销毁表；
8. 退出。

用户选择一个操作，则主函数调用相应的算法，完成操作。

**四、实验要求**

撰写实验报告，并将黑体、下划线的算法代码、运行结果写入实验报告；

顺序表

#include<stdio.h>  
#include<stdlib.h>  
  
typedef struct{  
    int \*elem;  
    int length;  
    int listsize;  
}SqList;  
  
bool InitList(SqList &L){  
    int LIST\_INIT\_SIZE;  
    printf("请输入表最大长度:");  
    scanf("%d",&LIST\_INIT\_SIZE);  
    L.elem = (int \*)malloc(LIST\_INIT\_SIZE\*4);  
    if(!L.elem)exit(EOVERFLOW);  
    L.length = 0;  
    L.listsize = LIST\_INIT\_SIZE;  
    return true;  
}  
  
//在第i个元素前插入新元素   1<=i<=len+1  
bool ListInsert(SqList &L,int i,int e){  
    for(int j=L.length-1;j>=i-1;--j){  
        L.elem[j+1] = L.elem[j];  
    }  
    L.elem[i-1] = e;  
    ++L.length;  
    return true;  
}  
  
bool ListDelete(SqList &L,int i){  
    if((i<1)||(i>L.length))return 0;  
    int\* p = &(L.elem[i-1]);  
    int \*q = L.elem + L.length-1;  
    for(++p;p<=q;++p){  
        \*(p-1) = \*p;  
    }  
    --L.length;  
    return 1;  
}  
  
void reverse(SqList &L){  
    int left = 0;  
    int right = L.length-1;  
    while(left < right){  
        int temp = L.elem[left];  
        L.elem[left] = L.elem[right];  
        L.elem[right] = temp;  
        left++;  
        right--;  
    }  
}  
  
void DestroyList(SqList &L) {  
    if (L.elem != NULL) {  
        free(L.elem);  // 释放存储元素的内存  
        L.elem = NULL; // 将指针置为 NULL，以避免野指针  
    }  
    L.length = 0;    // 重置顺序表的长度  
    L.listsize = 0;  // 重置顺序表的总容量  
}  
  
void lookup(SqList L,int e,int arr[]){  
    int count = 0;  
    for(int i=0;i<L.length;++i){  
        if(L.elem[i]==e){  
            arr[count] = i+1;  
            count++;  
        }  
    }  
}  
  
bool Empty(SqList L){  
    if(L.length==0){  
        return 1;  
    }else{  
        return 0;  
    }  
}  
void print(SqList L){  
    for(int i=0;i<L.length;++i){  
        printf("%d   ",L.elem[i]);  
    }  
}  
  
  
int main(){  
    int choose=0;  
    printf("请选择您要进行的操作：1.初始化顺序表 2.插入 3.删除值 4.逆序 5.是否为空 6.打印 7.查找元素 8.销毁表");  
    while(scanf("%d",&choose)){  
        switch (choose) {  
        case 1:  
            SqList S1;  
            InitList(S1);  
            break;  
        case 2:  
            int i,e1;  
            printf("请输入在第几个前添加和值");  
            scanf("%d %d",&i,&e1);  
            ListInsert(S1,i,e1);  
            break;  
        case 3:  
            int d;  
            printf("请输入要删除第几个");  
            scanf("%d",&d);  
            ListDelete(S1,d);  
            break;  
        case 4:  
            reverse(S1);  
            break;      
        case 5:  
            printf("%d",Empty(S1));  
            break;  
        case 6:  
            print(S1);  
            break;  
        case 7:  
            int e3=0;  
            printf("输入你要查找元素的值，我会给你一个它位置的数组");  
            scanf("%d",&e3);  
            int arr[10]={};  
            lookup(S1,e3,arr);  
            for(int i=0;i<10;i++){  
                if(arr[i]){  
                    printf("%d--",arr[i]);  
                }  
            break;  
        }  
          
    }  
}  
}

链表

#include**<**stdio**.**h**>**#include**<**stdlib**.**h**>**#include **<**iostream**>**#include **<**string**>**#include **<**vector**>  
using** **namespace** **std;  
  
typedef** **struct** LNode**{**    **int** data**;**    **struct** LNode **\*** next**;  
}\*** LinkList**;***//创造一个节点，如果创造的是头节点，则data为0*LinkList createLinkList**(int** data**)** **{**    LinkList newLinkList **=** **(**LinkList**)**malloc**(sizeof(struct** LNode**));**    newLinkList**->**data **=** data**;**    newLinkList**->**next **=** NULL**;**    **return** newLinkList**;  
}  
  
void** InsertToEnd**(**LinkList L**,int** data**){**    LinkList NewNode **;**    NewNode **=** createLinkList**(**data**);**    LinkList P**;**   
    P **=** L**;**    **while(**P**->**next**!=**NULL**){**        P **=** P**->**next**;**    **}**    P**->**next **=** NewNode**;***//*    *cout* *<<* *P->next->data* *<<* *endl;***}  
  
void** InsertByIndex**(**LinkList L**,int** index**,int** e**){**    LinkList NewNode**;**    NewNode **=** createLinkList**(**e**);**    **for(int** i**=**0**;**i**<**index**-**1**;**i**++){**        L **=** L**->**next**;**    **}***//找到要插入的位置的原元素的前一个元素*    NewNode**->**next **=** L**->**next**;**    L**->**next **=** NewNode**;  
}  
  
void** InsertByValue**(**LinkList L**,int** value**,int** e**){**    **while(**L**->**next**!=**NULL**){**        **if(**L**->**data**==**value**){**            LinkList NewNode**;**            NewNode **=** createLinkList**(**e**);**            NewNode**->**next **=** L**->**next**;**            L**->**next **=** NewNode**;**   
            L **=** L**->**next**;**        **}***//嫌麻烦了，就放在这个值的元素的前面吧*        L **=** L**->**next**;**    **}  
  
}  
  
void** DeleteByIndex**(**LinkList L**,int** index**){**    **for(int** i **=**0**;**i**<**index**-**1**;++**i**){**        L **=** L**->**next**;**    **}***//移动到要删的元素的前驱*    LinkList temp **=** L**->**next**;**    L**->**next **=** L**->**next**->**next**;**    free**(**temp**);  
}  
  
int** ChooseHeadNode**(**vector**<**string**>** listNames**){**    **int** selection**;**    cout **<<** **"请选择要操作的表头(填序号)："** **<<** endl**;**    **for** **(**size\_t i **=** 0**;** i **<** listNames**.**size**();** i**++)** **{**        cout **<<** i **+** 1 **<<** **". "** **<<** listNames**[**i**]** **<<** endl**;**    **}**    cout **<<** **">>>> ";**    cin **>>** selection**;**    **return** selection**;  
}  
  
void** print**(**LinkList L**){**    LinkList current **=** L**;**    **while(**current **!=** NULL**){**        printf**("%d -> ",** current**->**data**);**        current **=** current**->**next**;**    **}** **;**    printf**("NULL**\n**");  
}  
  
void** LookUp**(**LinkList L**,int** value**,int** arr**[]){**    **int** count **=** 0**;**    **int** i**=**0**;**    **while(**L**->**next**!=**NULL**){**        L **=** L**->**next**;**        i**++;**        **if(**L**->**data**==**value**){**            arr**[**count**]=**i**;**            count**++;**        **}**    **};  
}**LinkList merge**(**LinkList L1**,** LinkList L2**)** **{**    *//* *创建新链表用于存储合并结果*    LinkList L3 **=** **(**LinkList**)**malloc**(sizeof(struct** LNode**));**    L3**->**data **=** 0**;**    L3**->**next **=** NULL**;**      
    LinkList current1 **=** L1**->**next**;**    LinkList current2 **=** L2**->**next**;**    LinkList current3 **=** L3**;**      
    *//* *遍历两个链表*    **while** **(**current1 **!=** NULL **&&** current2 **!=** NULL**)** **{**        **if** **(**current1**->**data **<=** current2**->**data**)** **{**            current3**->**next **=** createLinkList**(**current1**->**data**);**            current1 **=** current1**->**next**;**        **}** **else** **{**            current3**->**next **=** createLinkList**(**current2**->**data**);**            current2 **=** current2**->**next**;**        **}**        current3 **=** current3**->**next**;**    **}**      
    *//* *处理剩余部分*    **while** **(**current1 **!=** NULL**)** **{**        current3**->**next **=** createLinkList**(**current1**->**data**);**        current1 **=** current1**->**next**;**        current3 **=** current3**->**next**;**    **}**      
    **while** **(**current2 **!=** NULL**)** **{**        current3**->**next **=** createLinkList**(**current2**->**data**);**        current2 **=** current2**->**next**;**        current3 **=** current3**->**next**;**    **}**      
    **return** L3**;  
}  
  
int** main**(){**    **int** choose**=**0**;**    vector**<**LinkList**>** lists**;**  *//* *存储多个顺序表*    vector**<**string**>** listNames**;**  *//* *存储顺序表的名称*    printf**("请选择您要进行的操作：1.初始化链表，加入头节点 2.插入 3.删除值 4.打印 5.查找元素 6.销毁表 7.合并**\n**");**    string name**;**    **while(**scanf**("%d",&**choose**)){**        **switch** **(**choose**)** **{**        **case** 1**:**            cout **<<** **"请输入链表名称>> "<<**endl**;**            cin **>>** name**;**              
            LinkList HeadNode**;**            HeadNode **=** createLinkList**(**0**);**              
              
            **if** **(**HeadNode**)** **{**                lists**.**push\_back**(**HeadNode**);**                listNames**.**push\_back**(**name**);**                cout **<<** **"链表 "** **<<** name **<<** **" 创建成功"** **<<** endl**;**            **}** **else** **{**                cout **<<** **"链表创建失败"** **<<** endl**;**            **}**            **break;**      
        **case** 2**:**              
            **static** **int** selection **=** 0**;**            selection **=** ChooseHeadNode**(**listNames**);**            **if** **(**selection **>** 0 **&&** selection **<=** **static\_cast<int>(**lists**.**size**()))** **{**                cout **<<** **"请输入你想怎么插 1.末尾2.按序3.按值"** **<<** endl**;**                **int** how**;**                cin **>>** how**;**                **switch** **(**how**)** **{**                **case** 1**:**                    cout **<<** **"输入你想要添加的值"** **<<** endl**;**                    **int** data**;**                    cin **>>** data **;**                    InsertToEnd**(**lists**[**selection**-**1**],**data**);**                    **break;**                **case** 2**:**                    cout **<<** **"输入你想要添加的位置和值"** **<<** endl**;**                    **int** index**,**val1**;**                    cin **>>** index**>>**val1 **;**                    InsertByIndex**(**lists**[**selection**-**1**],**index**,**val1**);**                    **break;**                **default:**                    cout **<<** **"输入位置值和结点值"** **<<** endl**;**                    **int** val2**,**val3**;**                    cin **>>** val2**>>**val3 **;**                    InsertByValue**(**lists**[**selection**-**1**],**val2**,**val3**);**                    **break;**                **}**            **}** **else** **{**                cout **<<** **"无效的选择"** **<<** endl**;**            **}**            **break;**        **case** 3**:**            cout **<<** **"输入你想要删除的值的位置"** **<<** endl**;**            **int** index**;**            cin **>>** index **;**            DeleteByIndex**(**lists**[**selection**-**1**],**index**);**            **break;**        **case** 4**:**            selection **=** ChooseHeadNode**(**listNames**);**            **if** **(**selection **>** 0 **&&** selection **<=** **static\_cast<int>(**lists**.**size**()))** **{**                print**(**lists**[**selection**-**1**]);**            **}** **else** **{**                cout **<<** **"无效的选择"** **<<** endl**;**            **}**            **break;**        **case** 5**:**            selection **=** ChooseHeadNode**(**listNames**);**            **if** **(**selection **>** 0 **&&** selection **<=** **static\_cast<int>(**lists**.**size**()))** **{**                **int** arr**[**10**]={};**                **int** value**;**                cin **>>** value **;**                LookUp**(**lists**[**selection**-**1**],**value**,**arr**);**                **for(int** i**=**0**;**i**<**10**;**i**++){**                    **if(**arr**[**i**]){**                        printf**("%d--",**arr**[**i**]);**                    **}**                **}**            **}** **else** **{**                cout **<<** **"无效的选择"** **<<** endl**;**            **}**            **break;**        **case** 6**:**            selection **=** ChooseHeadNode**(**listNames**);**            **if** **(**selection **>** 0 **&&** selection **<=** **static\_cast<int>(**lists**.**size**()))** **{**                free**(**lists**[**selection**-**1**]);**                lists**.**erase**(**lists**.**begin**()-**1**+**selection**);**                listNames**.**erase**(**listNames**.**begin**()-**1**+**selection**);**            **}** **else** **{**                cout **<<** **"无效的选择"** **<<** endl**;**            **}**            **break;**        **case** 7**:**            **int** selection1 **=** ChooseHeadNode**(**listNames**);**            **int** selection2 **=** ChooseHeadNode**(**listNames**);**            **if** **(**selection1**>**0 **&&** selection2 **>** 0 **&&** selection1 **<=** **static\_cast<int>(**lists**.**size**())** **&&** selection2 **<=** **static\_cast<int>(**lists**.**size**()))** **{**                print**(**merge**(**lists**[**selection1**-**1**],**lists**[**selection2**-**1**]));**            **}** **else** **{**                cout **<<** **"无效的选择"** **<<** endl**;**            **}**            **break;**        **}**    **}  
}**