# 实验一 线性表的顺序存储系统维护

**一、实验目的**

1.掌握线性表的顺序存储的定义和基本使用方法。

2.掌握线性表的顺序存储存储单元的排列特点。

3.掌握线性表的顺序存储系统的建立、查找 、修改、插入、删除操作，学会相关的函数定义和调用。

**二、实验内容**

1.建立一个顺序表。

2.能够对建立的顺序表进行查找、修改、插入、删除等操作。当输入指令错误时，能够提示错误信息。主函数中可以选择由switch\case语句构成主菜单，再根据提示进行相应操作。

**三、实验指导**

1.在线性表的建立时，可直接用数组赋初值；

2.在查找功能中要实现的功能为：当找到时该值时返回该值所在节点，找不到时返回-1；

3.修改功能是在查找的基础上，将找到的值加以修改；

4.在插入功能中要实现的功能为：在找到指定节点后，当线性表满时，提示不能插入，当线性表不满时，插入数据；

5.删除功能主要实现：当线性表为空或者删除位置超出线性表长度时，都显示位置错误，其他情形进行删除操作。

**四、代码实现**

*//Experiment 1: Sequence List by:Yang Yujie using C++*#include <iostream>  
#define **MaxLength** 100  
#define **OK** 1  
#define **ERROR** 0  
#define **LengthInvalid** (-1)  
#define **OVERFLOW** (-2)  
  
using namespace std;  
typedef int Status;  
typedef struct SequenceList {  
 int value[**MaxLength**]={};  
 int length=0;  
} SequenceList;  
  
void ListInitialize (SequenceList &L);  
void ShowSequenceList (SequenceList L);  
Status LocateElem (SequenceList L, int GoalElem, int &Location);  
Status ModifyElem (SequenceList &L, int FormerElem, int NewElem);  
Status ListInsert (SequenceList &L, int Location, int InsertElem);  
Status ListDelete (SequenceList &L, int Location, int &ElementDelete);  
  
int main() {  
 int command;  
 SequenceList test;  
 cout << "Experiment 1: Sequence List." << endl  
 << "<Instruction> Please initialize the sequence list." << endl;  
 ListInitialize (test);  
 ShowSequenceList(test);  
 cout << endl  
 << "<Instruction> Please type in the command number to operate:"  
 << endl << endl;  
 cout << "/\* The command corresponds to operations.\n"  
 " \* command -1:Terminate the program.\n"  
 " \* command 1 :LocateElem.\n"  
 " \* command 2 :ModifyElem.\n"  
 " \* command 3 :ListInsert.\n"  
 " \* command 4 :ListDelete.\n"  
 " \*/" << endl;  
 while (cin >> command) {  
 switch (command) {  
 case 1: {  
 int GoalElem, location;  
 int flag1 = 1;  
 int ReturnValue1;  
 while (flag1) {  
 cout << "/\* LocateElem \*/" << endl;  
 ShowSequenceList(test);  
 cout << "<Instruction> Please type in the located element:";  
 cin >> GoalElem;  
 ReturnValue1=LocateElem(test, GoalElem, location);  
 if (ReturnValue1==**OK**) {  
 cout << "<Instruction> Succeed!" << endl  
 << "<Instruction> The position index of the element:"  
 << '<' << location << '>' << endl;  
 flag1 = 0;  
 } else {  
 cout << "<Instruction> Failed!" << endl << endl;  
 if (ReturnValue1==**LengthInvalid**) {  
 flag1 = 0;  
 }  
 }  
 }  
 } break;  
  
 case 2: {  
 int FormerElem, NewElem;  
 int flag2 = 1;  
 int ReturnValue2;  
 while (flag2) {  
 cout << "/\* ModifyElem \*/" << endl;  
 ShowSequenceList (test);  
 cout << "<Instruction> Please type in the former element:";  
 cin >> FormerElem;  
 cout << "<Instruction> Please type in the new element:";  
 cin >> NewElem;  
 ReturnValue2=ModifyElem(test, FormerElem, NewElem);  
 if (ReturnValue2==**OK**) {  
 cout << endl << "<Instruction> Succeed!" << endl  
 << "<Instruction> The former element "  
 << '<' << FormerElem << '>'  
 << " has been replaced by "  
 << '<' << NewElem << '>' << endl;  
 ShowSequenceList(test);  
 flag2 = 0;  
 } else {  
 cout << "<Instruction> Failed!" << endl << endl;  
 if(ReturnValue2==**LengthInvalid**){  
 flag2 = 0;  
 }  
 }  
 }  
 } break;  
  
 case 3: {  
 int InsertElem, Location;  
 int flag3 = 1;  
 int ReturnValue3;  
 while (flag3) {  
 cout << "/\* ListInsert \*/" << endl;  
 ShowSequenceList (test);  
 cout << "<Instruction> Please type in the inserted element:";  
 cin >> InsertElem;  
 cout << "<Instruction> Please type in the position index:";  
 cin >> Location;  
 ReturnValue3 = ListInsert(test, Location, InsertElem);  
 if (ReturnValue3==**OK**) {  
 cout << "<Instruction> Succeed!" << endl  
 << "The element "  
 << '<' << InsertElem << '>'  
 << " has been inserted to the position index "  
 << '<' << Location << '>' << endl;  
 ShowSequenceList(test);  
 flag3 = 0;  
 } else {  
 cout << "<Instruction> Failed!" << endl << endl;  
 if (ReturnValue3==**LengthInvalid**) {  
 flag3 = 0;  
 }  
 }  
 }  
 } break;  
  
 case 4: {  
 int LocationDelete;  
 int flag4 = 1;  
 int ElementDelete;  
 int ReturnValue4;  
 while (flag4) {  
 cout << "/\* ListDelete \*/" << endl;  
 ShowSequenceList (test);  
 cout << "<Instruction> Please type in the position index:";  
 cin >> LocationDelete;  
 ReturnValue4 = ListDelete(test, LocationDelete, ElementDelete);  
 if (ReturnValue4==**OK**) {  
 cout << "<Instruction> Succeed!" << endl  
 << "The element "  
 << '<' << ElementDelete << '>'  
 << " at the position index "  
 << '<' << LocationDelete << '>'  
 <<" has been deleted."<< endl;  
 ShowSequenceList(test);  
 flag4 = 0;  
 } else {  
 cout << "<Instruction> Failed!" << endl << endl;  
 if (ReturnValue4==**LengthInvalid**) {  
 flag4 = 0;  
 }  
 }  
 }  
 } break;  
  
 case -1: {  
 cout << " <Instruction> The program terminated! " << endl;  
 } break;  
  
 default: {  
 cout << "The command is invalid!" << endl;  
 } break;  
  
 }  
 if (command == -1) {  
 break;  
 }  
 cout << endl  
 << "<Instruction> Please type in the command number to operate."  
 << endl << endl;  
 cout << "/\* The command corresponds to operations.\n"  
 " \* command -1:Terminate the program.\n"  
 " \* command 1 :LocateElem.\n"  
 " \* command 2 :ModifyElem.\n"  
 " \* command 3 :ListInsert.\n"  
 " \* command 4 :ListDelete.\n"  
 " \*/" << endl;  
 }  
 return 0;  
}  
  
void ListInitialize (SequenceList &L) {  
 cout << "Input format:" << endl << "a1 a2 a3 ... an\\n" << endl;  
 int tempch;  
 for (int i=0; i<**MaxLength**; ++i) {  
 cin >> L.value[i];  
 L.length++;  
 tempch = getchar();  
 if (tempch == '\n') {  
 break;  
 }  
 }  
}  
  
void ShowSequenceList (SequenceList L) {  
 cout << "<ShowSequenceList> Sequence list : {";  
 for (int i=0; i<L.length; i++) {  
 cout << L.value[i];  
 if (i!=L.length-1) {  
 cout << ',';  
 }  
 }  
 cout << '}' << endl;  
 cout << "<ShowSequenceList> The length of the list: " << L.length << endl;  
}  
  
Status LocateElem (SequenceList L, int GoalElem, int &Location) {  
 for (int i=0; i<L.length; i++) {  
 if (GoalElem==L.value[i]) {  
 Location = i;  
 return **OK**;  
 }  
 }  
 if (L.length==0) {  
 return **LengthInvalid**;  
 }  
 return **ERROR**;  
}  
  
Status ModifyElem (SequenceList &L, int FormerElem, int NewElem) {  
 for (int i=0; i<L.length; i++) {  
 if (FormerElem==L.value[i]) {  
 L.value[i]=NewElem;  
 return **OK**;  
 }  
 }  
 if (L.length==0) {  
 return **LengthInvalid**;  
 }  
 return **ERROR**;  
}  
  
Status ListInsert (SequenceList &L, int Location, int InsertElem) {  
 if (Location<0 || Location>L.length && L.length!=**MaxLength**) {  
 cout << "<Instruction> The position is invalid." << endl;  
 return **OVERFLOW**;  
 } else if (L.length==**MaxLength**){  
 cout << "<Instruction> You cannot insert element." << endl;  
 return **LengthInvalid**;  
 } else {  
 for (int i=L.length; i>=Location; i--) {  
 L.value[i+1]=L.value[i];  
 }  
 L.value[Location]=InsertElem;  
 L.length++;  
 return true;  
 }  
}  
  
Status ListDelete (SequenceList &L, int Location, int &ElementDelete) {  
 if (Location<0 || Location>L.length-1 && L.length!=0) {  
 cout << "<Instruction> The position is invalid." << endl;  
 return **OVERFLOW**;  
 } else if (L.length==0) {  
 cout << "<Instruction> You cannot delete element." << endl;  
 return **LengthInvalid**;  
 } else {  
 ElementDelete = L.value[Location];  
 for (int i=Location+1; i<L.length; i++) {  
 L.value[i-1]=L.value[i];  
 }  
 L.length--;  
 return **OK**;  
 }  
}

**五、程序调试**

注：程序中控制台输出的结果中，“位置(Location)”均表示指定元素在数组的下标。

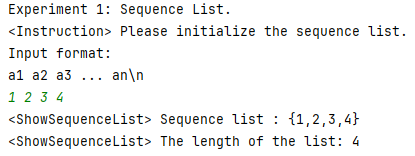


图 1 顺序表初始化

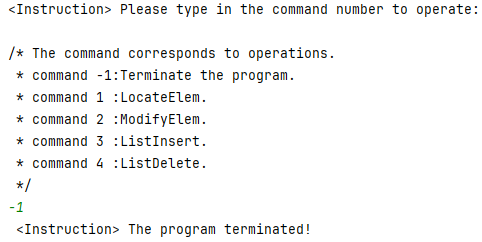


图 2 程序退出

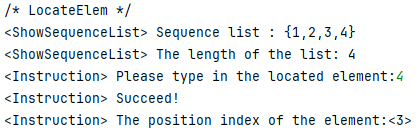


图 3 顺序表查找成功

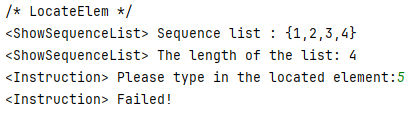


图 4 顺序表查找失败

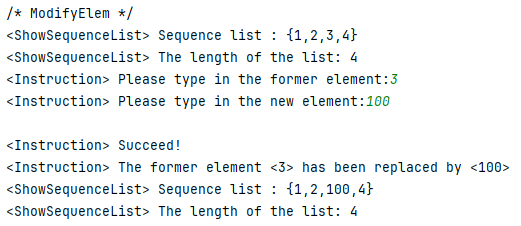


图 5 顺序表修改成功

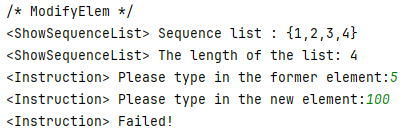


图 6 顺序表修改失败

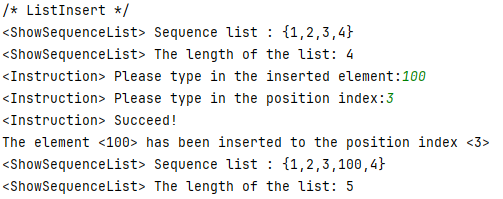


图 7 顺序表插入成功

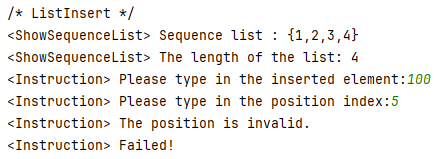


图 8 顺序表插入失败

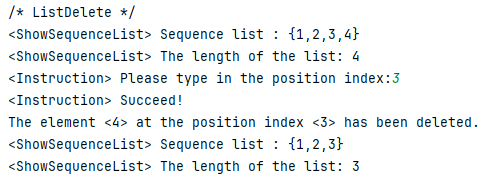


图 9 顺序表删除成功

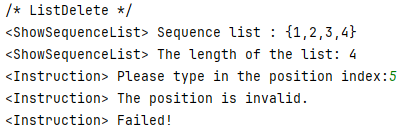


图 10 顺序表删除失败