

《数据结构》

实验报告

姓名： 庄李晨

学号： 10522238

成绩：

电子与计算机工程学院

School of Electronic & Computer Engineering

2023年9月

【实验名称】单链表的实现

【实验目的】

1. 掌握单链表的基本操作，实现单链表的插入、删除、查找等基本运算。
2. 运用单链表实现简单应用。

【实验内容】

1. 实验要求：

1、仔细阅读实验代码，补充下面9个函数的实现过程。

（1）用尾插法构造单链表，新结点插入在单链表尾结点之后，用户依次输入n个数据元素

（2）用头插法构造单链表，新结点插入在单链表头结点之后，用户依次输入n个数据元素

（3）返回单链表中第i个结点的值

（4）在单链表i位置插入元素e

（5）删除单链表中位于i位置的结点，成功输出"已删除第i个元素"，失败则输出原因（链表为空，或者未找到第i个元素 )

（6）清空单链表

（7）遍历单链表 ，输出各个结点的值

（8）销毁单链表

（9）删除单链表中值为e的结点，成功输出"已删除e元素"，失败则输出原因（链表为空，或者未找到e元素）。

2、补充完成main（）函数的定义，调试单链表程序，给出测试用例及测试结果。

3、（选作）构造有序单链表，初始时单链表为空，将元素按照值大小，从小到大插入到链表中,例如用户依次输入数据元素(5,7,2,3,6),单链表中结点顺序为 （2，3，5，6，7）。算法提示：需要按照元素值查找插入点位置，然后插入新结点。

4、（选作）采用单链表存储结构，实现一元多项式的定义及基本操作。

1. 程序清单

#include<iostream>

using namespace std;

typedef int ElemType;

typedef int Status;

std::unique\_ptr<int> Ll(new int(10));

// 结点定义

typedef struct LNode

{

ElemType data; // 数据域

struct LNode\* next; // 指针域

} LNode, \* LinkList;

//1.用尾插法构造单链表，新结点插入在单链表尾结点之后

//用户依次输入n个数据元素

void CreateListTail(LinkList& L, int n)

{

int x;

LinkList p;

L = (LinkList)malloc(sizeof(LNode));

L->next = NULL;

LinkList tail = L; // 新加的尾节点指针

cout << "依次输入元素" << endl;

for (x = n; x > 0; --x)

{

p = (LinkList)malloc(sizeof(LNode));

cin >> p->data;

p->next = NULL; // 新节点的 next 设置为 NULL

tail->next = p; // 将新节点链接到尾节点后面

tail = p; // 更新尾节点的指针

}

}//CreateList

//2.用头插法构造单链表，新结点插入在单链表头结点之后

//用户依次输入n个数据元素

void CreateListHead(LinkList& L, int n)

{

LinkList p;

L = (LinkList)malloc(sizeof(LNode));

L->next = NULL;//先建立一个带头节点的单链表

cout << "请输入单链表元素：" << endl;

for (int x = n; x > 0; --x)

{

p = (LinkList)malloc(sizeof(LNode));

cin >> p->data;

p->next = L->next;

L->next = p;

}

}

//3.返回单链表中第i个结点的值

Status GetElem(LinkList& L, int i)

{

int x2 = 1;

LNode\* p = L->next;

while (i > x2)

{

p = p->next;

x2++;

}

if (i == x2)

{

cout << "第i个结点的值为：" << endl;

cout << p->data << endl;

}

return 1;

}

//4.在单链表i位置插入元素e

Status ListInsert(LinkList& L, int i, ElemType e)

{

LinkList pre;//前驱结点

pre = L;

int x3 = 0;

for (x3 = 1; x3 < i; x3++)

{

pre = pre->next;

}//查找i的位置

LNode\* p;//插入的结点

p = (LinkList)malloc(sizeof(LNode));

p->data = e;

p->next = pre->next;

pre->next = p;

return 1;

}

//5，删除单链表中位于i位置的结点，成功输出"已删除第i个元素"，

//失败则输出原因（链表为空，或者未找到第i个元素 )

void ListDelete(LinkList& L, int i)

{

LinkList p, q;

int x4 = 2;

p = L->next;

q = (LinkList)malloc(sizeof(LNode));

while (p->next && x4 < i) //查找i的位置

{

p = p->next;

++x4;

}

if (!(p->next) || x4 > i)

{

cout << "第i个元素不存在" << endl;

}

else

q = p->next;

p->next = q->next;//q的后继赋值给p的后继

delete(q);//释放（删除）q

cout << "已经成功删除第" << x4 << "个结点的元素" << endl;

}

//6.清空单链表

void ClearList(LinkList& L)

{

LNode\* p;

while (L)

{

p = L;

L = L->next;

delete(p);

}

}

//7.遍历单链表 ，输出各个结点的值

void Traverse(LinkList L)

{

cout << "单链表的数据为:";

LNode\* p = L->next;// 从第一个节点开始遍历，跳过头节点

while (p != NULL)

{

cout << p->data << ' ';// 输出节点的值

p = p->next;// 移动到下一个节点

}

cout << endl;

}

//8.销毁单链表

void DestroyList(LinkList& L)

{

L = (LinkList)malloc(sizeof(LNode));

L->next = NULL;

}

//9.删除单链表中值为e的结点，成功输出"已删除e元素"，

//失败则输出原因（链表为空，或者未找到e元素）

void DeleteElem(LinkList& L, ElemType e)

{

LNode\* p, \* pre=NULL;

p = L->next;

while (p != nullptr && p->data != e)

{

pre = p;

p = p->next;

}

if (!p)

{

cout << "链表为空" << endl;

}

else if (p->data != e)

{

cout << "单链表不存在元素" << e << endl;

}

else if (p->data == e)

{

pre->next = p->next;

delete(p);

}

}

int main(void)

{

LinkList Ll;

int n = 0;

int c = 0;

while (c != 9)

{

cout << endl << "1. 尾插法创建单链表";

cout << endl << "2. 头插法创建单链表";

cout << endl << "3. 在单链表中查找第i个元素";

cout << endl << "4. 在单链表中第i个位置插入元素";

cout << endl << "5. 在单链表中删除第i个元素";

cout << endl << "6. 在单链表中删除e元素";

cout << endl << "7. 遍历单链表";

cout << endl << "8. 销毁单链表";

cout << endl << "9. 退出";

cin >> c;

switch (c)

{

case 1:

cout << "请输入链表的元素个数" << endl;

int x;

cin >> x;

CreateListTail(Ll, x);

Traverse(Ll);

break;

case 2:

cout << "请输入链表个数的个数" << endl;

int x1;

cin >> x1;

CreateListHead(Ll, x1);

Traverse(Ll);

break;

case 3:

int i;

cout << "查询第i个结点的值" << endl;

cin >> i;

GetElem(Ll, i);

break;

case 4:

int e;

int x3;

cout << "请输入要插入的位置：" << endl;

cin >> x3;

cout << "请输入要插入的元素" << endl;

cin >> e;

if (ListInsert(Ll, x3, e))

{

cout << "插入成功";

Traverse(Ll);

}

else

{

cout << "插入失败";

Traverse(Ll);

}

break;

case 5:

int x4;

cout << "请输入要删除的元素的位序" << endl;

cin >> x4;

ListDelete(Ll, x4);

Traverse(Ll);

break;

case 6:

cout << "请输入要删除的元素：" << endl;

cin >> e;

DeleteElem(Ll, e);

Traverse(Ll);

break;

case 7:

cout << "单链表遍历成功" << endl;

//Traverse(Ll);

break;

case 8:

DestroyList(Ll);

cout << "销毁单链表成功" << endl;

case 9:

break;

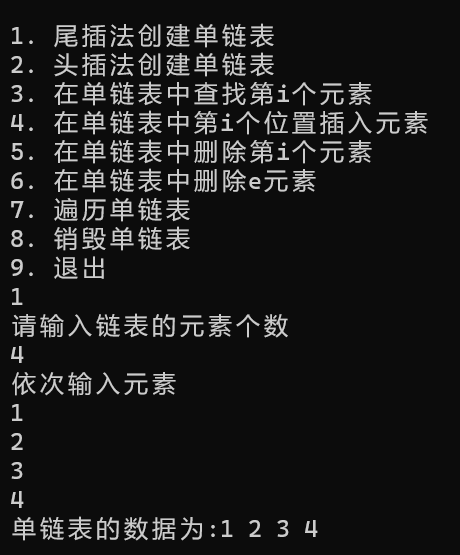
}

}

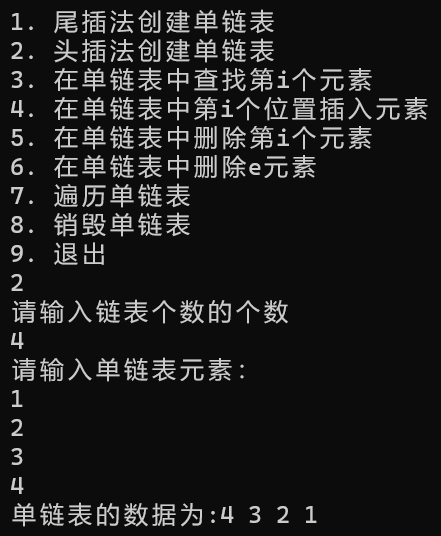
}

1. 结果截图

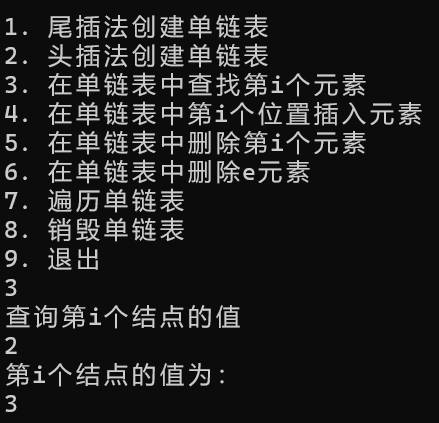
1.尾插法

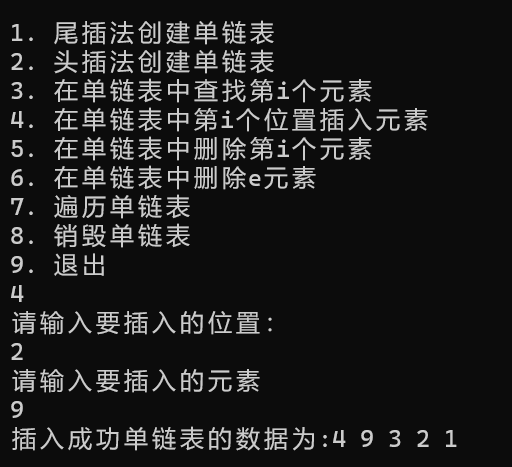


2.头插法

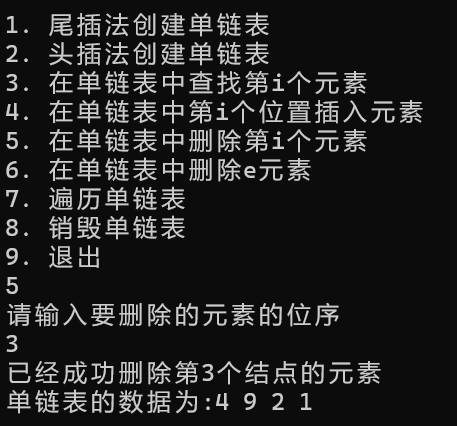


1. 查找i

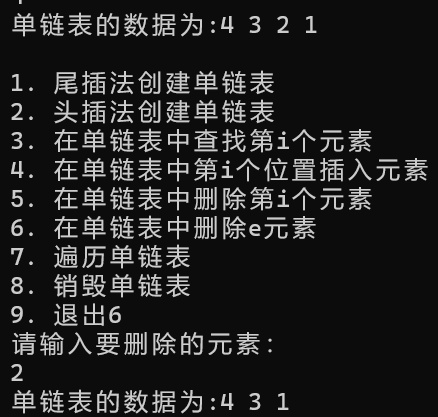


4.在i位置插入元素

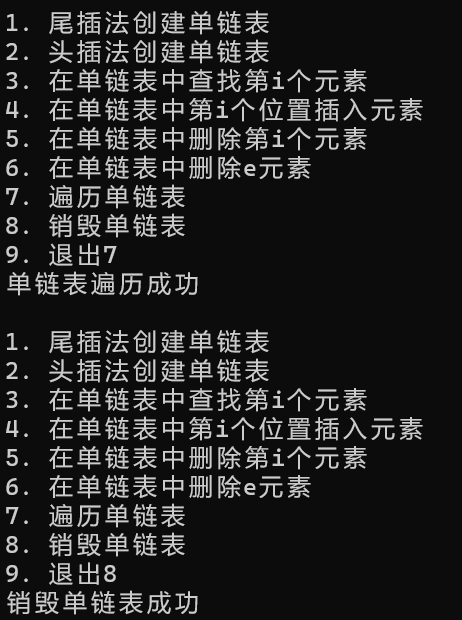
5.删除第i个元素



1. 删除e



1. 遍历单链表 8.销毁单链表



【实验心得】：通过本次学习单链表的操作，了解了头插法，尾插法等主要的经典算法，还将指针的应用重新复习与新学习，加强了C++方面的学习。