西安财经学院 信息学院

姓名：程驰

学号：1931054041

班级：计本1902

指导教师：李薇

成绩：

数据结构 实验报告

实验名称 顺序表的基本操作

实验日期：2020年10月02日

**一、实验目的：**

1、复习C语言程序设计中的知识。

2、掌握线性表的链式存储结构的表示和实现方法。

3、掌握单链表基本操作的算法实现。

**二、实验内容：**

1．建立单链表。

2．在单链表上实现插入、删除和查找等操作。

三、实验要求：

编写实现单链表的基本算法（初始化、查找、插入、删除等）的函数，并在此基础上设计一个主程序完成如下功能：

1. 初始化单链表H,类型自选;

⑵分别采用头插法和尾插法建立单链表H，如（a,b,c,d,c）；

⑶输出单链表H的长度；

⑷输出单链表H的第i个元素，如输出单链表H的第3个元素；

⑸输出给定元素的位置，如输出元素a的位置；

⑹在第i个元素前插入给定元素，如在第4个元素前插入元素f；

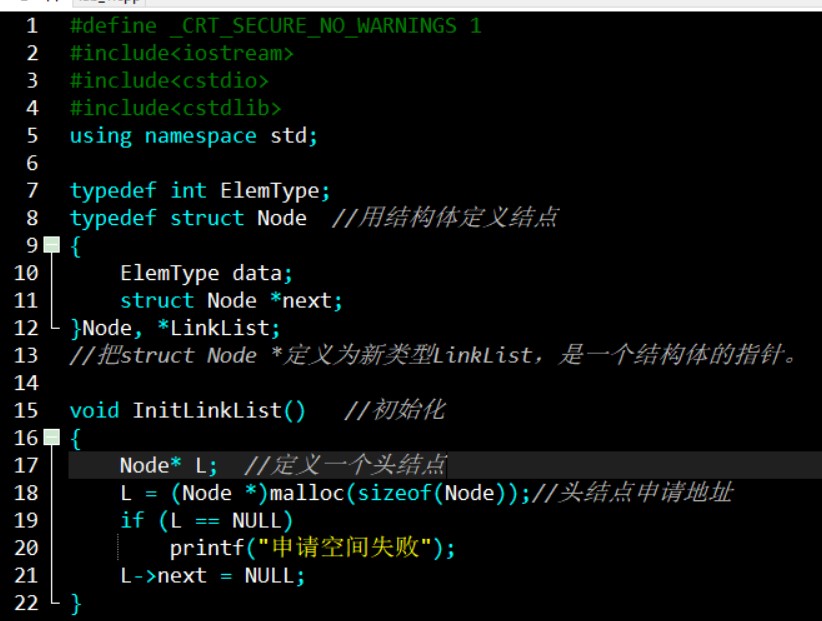
⑺删除单链表H的第i个元素，如删除单链表H的第3个元素。

⑻输出单链表

四、实验步骤：

1.确定实验环境：中文window10 系统、Visual Studio 2019，Dev-C++

2.单链表的初始化



3.头插法尾插法建表

LinkList CreateLinkedListHead(int n)//头插法

{

int i;

int x;

Node \*L;

L = (Node \*)malloc(sizeof(Node));

L->next = NULL;

printf("输入元素：");

for (i = 0; i<n; i++)

{

Node \*p; //要插入的结点

p = (Node \*)malloc(sizeof(Node));

scanf("%d", &x);

p->data = x;

p->next = L->next;

L->next = p;

}

return L;

}

LinkList CreateLinkedListTail(int n)//尾插法

{

int x;

Node \*L;

L = (Node \*)malloc(sizeof(Node));

L->next = NULL;

Node \*r;

r = L;

printf("输入元素:");

while (n--)

{

scanf("%d", &x);

Node \*p;

p = (Node \*)malloc(sizeof(Node));

p->data = x;

r->next = p;

r = p;//之后的循环遇到r就是等同于p r->next就是p->next

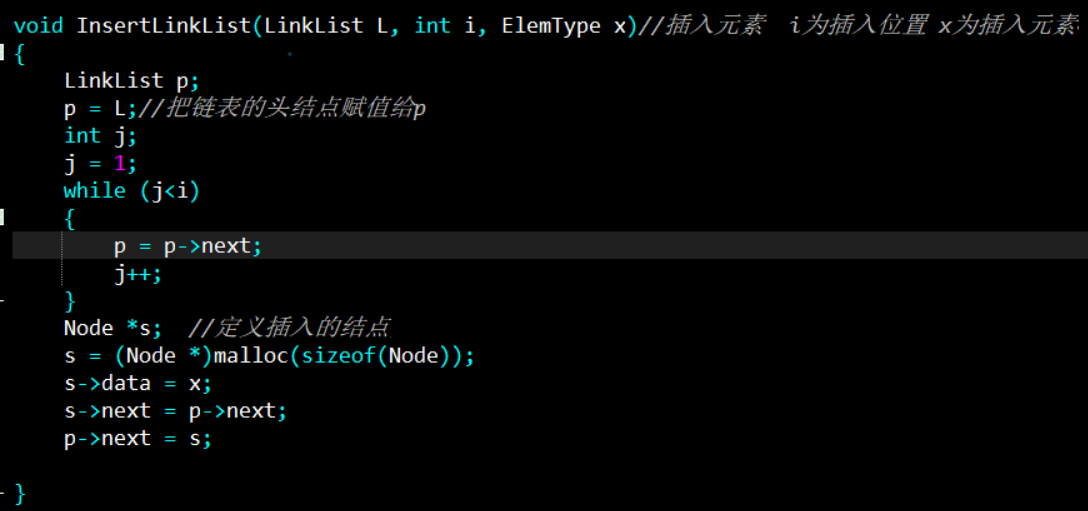
}

r->next = NULL;

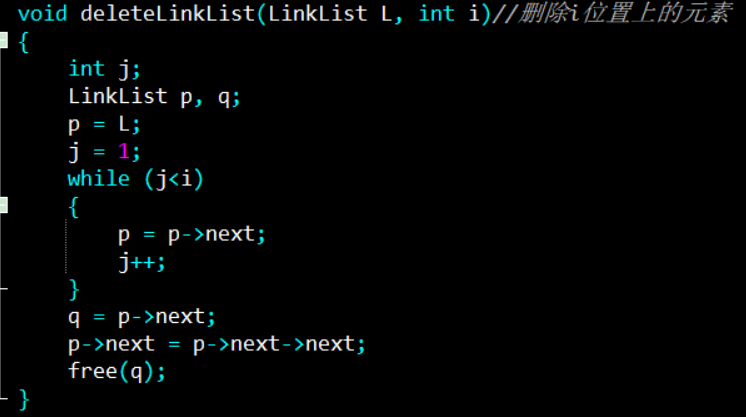
return L;

}

4.指定位置插入元素



5.删除操作



6.按位置获取元素和获取元素的位置

int GetElem(LinkList L, int i)//获取元素

{

ElemType e;

int j;

LinkList p;

p = L->next;

j = 1;

while (p && j<i)

{

p = p->next;

j++;

}

if (!p || j > i)

{

printf("第%d个节点不存在\n", i);

return 0;

}

else

e = p->data;

return e;

}

void ListLocate\_L(LinkList L, ElemType e)//获取元素位置

{

Node \*p = L ;

int k=0;

while(p->next){

if(p->data==e){

printf("该元素的位置为%d\n",k);

return;

}

else{

p=p->next;

k++;

}

if(p==L){

printf("未找到该元素！\n");

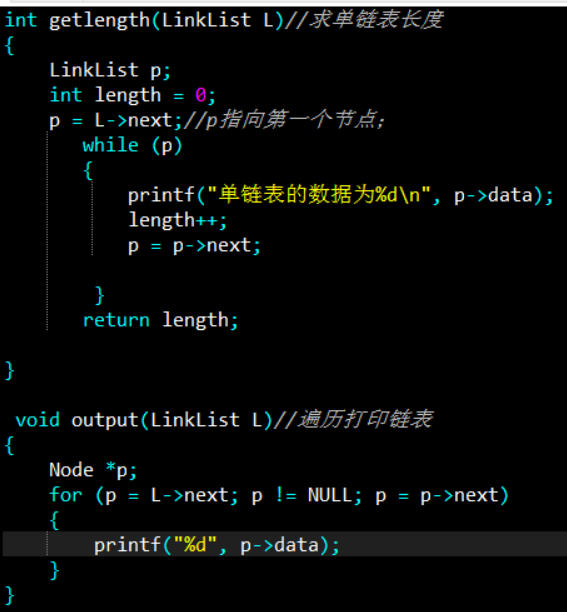
return;

}

}

}

7.求长度和打印打印单链表



附：程序清单

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

typedef int ElemType;

typedef struct Node //用结构体定义结点

{

ElemType data;

struct Node \*next;

}Node, \*LinkList;

//把struct Node \*定义为新类型LinkList，是一个结构体的指针。

void InitLinkList() //初始化

{

Node\* L; //定义一个头结点

L = (Node \*)malloc(sizeof(Node));//头结点申请地址

if (L == NULL)

printf("申请空间失败");

L->next = NULL;

}

LinkList CreateLinkedListHead(int n)//头插法

{

int i;

int x;

Node \*L;

L = (Node \*)malloc(sizeof(Node));

L->next = NULL;

printf("输入元素：");

for (i = 0; i<n; i++)

{

Node \*p; //要插入的结点

p = (Node \*)malloc(sizeof(Node));

scanf("%d", &x);

p->data = x;

p->next = L->next;

L->next = p;

}

return L;

}

LinkList CreateLinkedListTail(int n)//尾插法

{

int x;

Node \*L;

L = (Node \*)malloc(sizeof(Node));

L->next = NULL;

Node \*r;

r = L;

printf("输入元素:");

while (n--)

{

scanf("%d", &x);

Node \*p;

p = (Node \*)malloc(sizeof(Node));

p->data = x;

r->next = p;

r = p;//之后的循环遇到r就是等同于p r->next就是p->next

}

r->next = NULL;

return L;

}

void InsertLinkList(LinkList L, int i, ElemType x)//插入元素 i为插入位置 x为插入元素

{

LinkList p;

p = L;//把链表的头结点赋值给p

int j;

j = 1;

while (j<i)

{

p = p->next;

j++;

}

Node \*s; //定义插入的结点

s = (Node \*)malloc(sizeof(Node));

s->data = x;

s->next = p->next;

p->next = s;

}

void deleteLinkList(LinkList L, int i)//删除i位置上的元素

{

int j;

LinkList p, q;

p = L;

j = 1;

while (j<i)

{

p = p->next;

j++;

}

q = p->next;

p->next = p->next->next;

free(q);

}

int GetElem(LinkList L, int i)//获取元素

{

ElemType e;

int j;

LinkList p;

p = L->next;

j = 1;

while (p && j<i)

{

p = p->next;

j++;

}

if (!p || j > i)

{

printf("第%d个节点不存在\n", i);

return 0;

}

else

e = p->data;

return e;

}

void ListLocate\_L(LinkList L, ElemType e)//获取元素位置

{

Node \*p = L ;

int k=0;

while(p->next){

if(p->data==e){

printf("该元素的位置为%d\n",k);

return;

}

else{

p=p->next;

k++;

}

if(p==L){

printf("未找到该元素！\n");

return;

}

}

}

int getlength(LinkList L)//求单链表长度

{

LinkList p;

int length = 0;

p = L->next;//p指向第一个节点；

while (p)

{

printf("单链表的数据为%d\n", p->data);

length++;

p = p->next;

}

return length;

}

void output(LinkList L)//遍历打印链表

{

Node \*p;

for (p = L->next; p != NULL; p = p->next)

{

printf("%d", p->data);

}

}

int main()

{

int n;

int i;

int x;

LinkList H;

printf("输入单链表的长度（头插法）:");

scanf("%d", &n);

H = CreateLinkedListHead(n);

output(H);

printf("\n");

printf("输入单链表的长度（尾插法）:");

scanf("%d", &n);

H = CreateLinkedListTail(n);

output(H);

printf("\n");

printf("单链表长度为%d", getlength(H));

printf("\n");

printf("输入要获取哪一位置的元素：");

scanf("%d", &i);

printf("%d\n", GetElem(H, i));

printf("输入要获取哪一元素的位置：");

scanf("%d", &i);

ListLocate\_L(H,i);

printf("\n");

printf("输入要插入的元素e与位置i：");

scanf("%d %d", &x, &i);

InsertLinkList(H, i, x);

output(H);

printf("\n");

printf("输入要删除哪一位置上的元素：");

scanf("%d", &i);

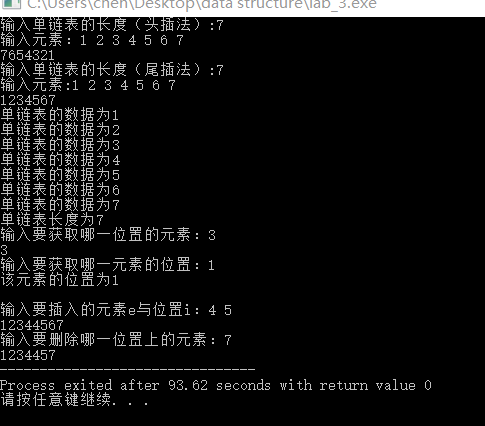
deleteLinkList(H, i);

output(H);

return 0;

}

五、实验结果：



六、实验总结：

通过这次实验，我对头插法和尾插法两种建立单链表的方式理解更加深入，编程能力和设计算法的能力有了显著提高。

在程序设计中还有很多不完美的地方，比如对异常情况的处理不够完善，这些是我需要重点关注的地方。