# 实验 链表

樊子谦

软件学院 软件工程 2024级1班

**任务一：**

1．实验要求

编程实现如下功能

（1）根据输入的一系列整数，以0标志结束，用***头插法***建立单链表，并输出单链表中各元素值，观察输入的内容与输出的内容是否一致。

（2）根据输入的一系列整数，以0标志结束，用***尾插法***建立单链表，并输出单链表中各元素值，观察输入的内容与输出的内容是否一致。

（3）在单链表的第i个元素之后插入一个值为x的元素，并输出插入后的单链表中各元素值。

（4）删除单链表中第i个元素，并输出删除后的单链表中各元素值。

（5）在单链表中查找第i个元素，如果查找成功，则显示该元素的值，否则显示该元素不存在。

头插法相当于逆序插入，尾插法是顺序插入。所以头插法输出的内容会颠倒。

插入和删除元素直接断链、连接节点即可。

代码如下：

struct node {

int val;

node\* nxt;

node(int x) : val(x), nxt(NULL) {}

node(int x, node\* nxt) : val(x), nxt(nxt) {}

node\* create\_list\_by\_head\_insert() {

node\* head = NULL;

int x;

while (cin >> x) {

if (x == 0) break;

node\* new\_node = new node(x, head);

head = new\_node;

}

return head;

}

node\* create\_list\_by\_tail\_insert() {

node \*head = NULL, \*tail = NULL;

int x;

while (cin >> x) {

if (x == 0) break;

node\* new\_node = new node(x);

if (head == NULL) {

head = tail = new\_node;

} else {

tail->nxt = new\_node;

tail = new\_node;

}

}

return head;

}

void output\_list(node\* head) {

node\* tmp = head;

while (tmp != NULL) {

cout << tmp->val << " ";

tmp = tmp->nxt;

}

cout << endl;

}

void insert\_after\_ith\_number(node\*& head, int i, int x) {

node\* tmp = head;

if (i == 0) {

node\* new\_node = new node(x, head);

head = new\_node;

return;

}

while (i > 1) {

tmp = tmp->nxt;

i--;

if (tmp == NULL) {

cout << "i is too large" << endl;

return;

}

}

node\* new\_node = new node(x, tmp->nxt);

tmp->nxt = new\_node;

}

void erase\_ith\_number(node\*& head, int i) {

node\* tmp = head;

if (i == 1) {

head = head->nxt;

delete tmp;

return;

}

while (i > 2) {

tmp = tmp->nxt;

i--;

if (tmp == NULL) {

cout << "i is too large" << endl;

return;

}

}

node\* del\_node = tmp->nxt;

tmp->nxt = del\_node->nxt;

delete del\_node;

}

void find\_ith\_number(node\* head, int i) {

node\* tmp = head;

for (int j = 1; j < i; j++) {

tmp = tmp->nxt;

if (tmp == NULL) {

cout << "The " << i << "th number is not exist." << endl;

return;

}

}

cout << "The " << i << "th number is " << tmp->val << endl;

}

void insertion\_sort(node\* head) {

for (node\* i = head->nxt; i != NULL; i = i->nxt) {

node\* t = head;

while (t != i) {

if (t->val > i->val) swap(t->val, i->val);

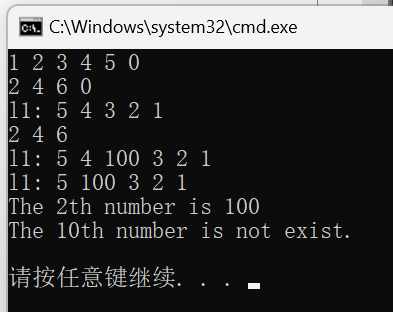
t = t->nxt;

}

}

}

};



**任务二：**

在实验任务一的基础上，实现插入排序

采用in-place的方法，在struct里面添加代码如下：

void insertion\_sort(node\* head) {

for (node\* i = head->nxt; i != NULL; i = i->nxt) {

node\* t = head;

while (t != i) {

if (t->val > i->val) swap(t->val, i->val);

t = t->nxt;

}

}

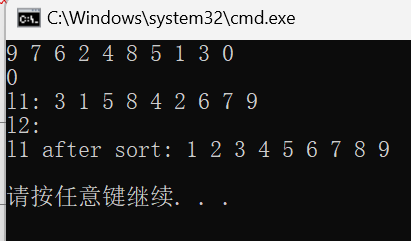
}

main函数添加代码：

l1->insertion\_sort(l1);

cout << "l1 after sort: ";

l1->output\_list(l1);



采用断链和连接的方式进行插入排序：（这个好难调试，直接断链的方法调了两小时还是不对，只能退而求其次用复杂度劣质的方法）

void insertion\_sort\_by\_node(node\* head) {

node\* i;

int cnt = 0;

for (i = head; i != NULL; i = i->nxt) {

cnt++;

int insert\_after\_ith;

node\* tmp = head;

for (insert\_after\_ith = 0; i->val > tmp->val; insert\_after\_ith++) {

tmp = tmp->nxt;

}

cout << "insert\_after\_ith: " << insert\_after\_ith << endl;

int val = i->val;

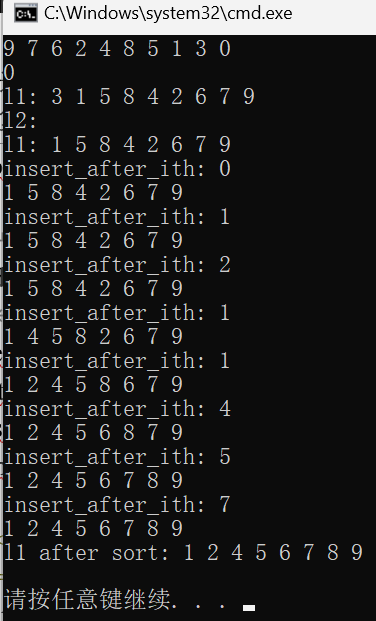
erase\_ith\_number(head, cnt);

insert\_after\_ith\_number(head, insert\_after\_ith, val);

output\_list(head);

}

}



因为不断有修改表头的操作，所以写代码的时候得非常小心，记得所有传参的head都要写成引用。写了这一版的代码才发现刚刚调的那2小时其中一个bug是因为忘记引用head了。悔恨。

**任务三：**

1.实验要求

（1）输入链表的长度和各元素的值，用尾插法建立双向循环链表，并输出链表中各元素值，观察输入的内容与输出的内容是否一致。

（2）在双向循环链表的第i个元素之前/后插入一个值为x的元素，并输出插入后的链表中各元素值。

（3）删除双向循环链表中第i个元素，并输出删除后的链表中各元素值。

（4）在双向循环链表中查找值为x元素，如果查找成功，则显示该元素在链表中的位置，否则显示该元素不存在。

2, 对比单链表核心算法提示

双向循环链表采用的存储结构描述如下：

         struct DULNODE

       { Elemtype data;  /\*数据域\*/

         struct DULNODE \*prior; /\*指向前驱结点的指针域\*/

         struct DULNODE \*next;/\*指向后继结点的指针域\*/

       } DulNODE,\*DuLinklist;

不论是建立双向循环链表还是在双向循环链表中进行插入、删除和查找操作，其操作方法和步骤都跟单链表类似。只不过要注意两点：

（1）凡是在操作中遇到有修改链的地方，都要进行前驱和后继两个指针的修改。

（2）单链表操作算法中的判断条件：p= =NULL 或p!=NULL ,在循环链表的操作算法中则需改为：p!= L,其中L为链表的头指针。

这里的“在循环链表的操作算法中则需改为：p!= L,其中L为链表的头指针”我并没有实施，而是直接用了链表长度n（偷懒）。完整代码如下：

#include <iostream>

using namespace std;

int n;

struct node {

int val;

node \*prev, \*nxt;

node(int x) : val(x), prev(NULL), nxt(NULL) {}

node(int x, node\* prev, node\* nxt) : val(x), prev(prev), nxt(nxt) {}

node\* create\_list() {

cin >> n;

node \*head = NULL, \*tail = NULL;

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

int x;

cin >> x;

if (head == NULL) {

head = new node(x);

tail = head;

} else {

tail->nxt = new node(x, tail, NULL);

tail = tail->nxt;

}

}

tail->nxt = head;

head->prev = tail;

return head;

}

void insert(node\*& head, node\*& tail, int i, bool front, int val) { // front==1:insert to the front of ith element

if (front) {

if (i == 1) {

node\* new\_node = new node(val, tail, head);

head->prev = new\_node;

head = new\_node;

} else {

node\* cur = head;

for (int j = 1; j < i; j++) cur = cur->nxt;

node\* new\_node = new node(val, cur->prev, cur);

cur->prev->nxt = new\_node;

cur->prev = new\_node;

}

} else {

if (i == n) {

node\* new\_node = new node(val, tail, head);

tail->nxt = new\_node;

tail = new\_node;

} else {

node\* cur = head;

for (int j = 1; j < i; j++) cur = cur->nxt;

node\* new\_node = new node(val, cur, cur->nxt);

cur->nxt->prev = new\_node;

cur->nxt = new\_node;

}

}

n++;

}

void erase\_ith\_element(node\*& head, node\*& tail, int i) {

if (i == 1) {

head = head->nxt;

head->prev = tail;

tail->nxt = head;

} else {

node\* cur = head;

for (int j = 1; j < i; j++) cur = cur->nxt;

cur->prev->nxt = cur->nxt;

cur->nxt->prev = cur->prev;

}

n--;

}

void find\_x(node\* head, int x) {

bool flag = 0;

node\* cur = head;

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

if (cur->val == x) {

cout << i + 1 << " ";

flag = 1;

}

cur = cur->nxt;

}

if (!flag) cout << "No this element!";

cout << endl;

}

void output\_list(const node\* head) {

const node\* cur = head;

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

cout << cur->val << " ";

cur = cur->nxt;

}

cout << endl;

}

};

int main() {

node\* l1 = NULL;

l1 = l1->create\_list();

l1->output\_list(l1);

int i, x;

string s;

cout << "Please input the position, value and the \"front\" or \"back\" of the element you want to insert:" << endl;

cin >> i >> x >> s;

l1->insert(l1, l1->prev, i, s == "front", x);

l1->output\_list(l1);

cout << "Please input the position, value and the \"front\" or \"back\" of the element you want to insert:" << endl;

cin >> i >> x >> s;

l1->insert(l1, l1->prev, i, s == "front", x);

l1->output\_list(l1);

cout << "Please input the position of the element you want to erase:" << endl;

cin >> i;

l1->erase\_ith\_element(l1, l1->prev, i);

l1->output\_list(l1);

cout << "Please input the element you want to find:" << endl;

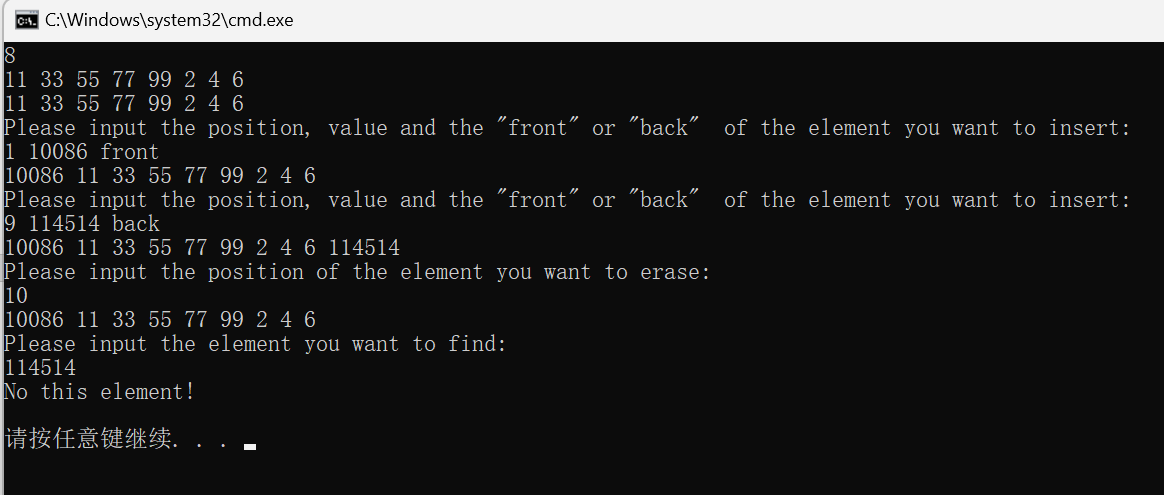
cin >> x;

l1->find\_x(l1, x);

return 0;

}

运行结果如下。运行正常。



心得：

在这次实验中，我深入学习了链表的基本概念和操作，包括单链表和双向循环链表的创建、插入、删除和查找等操作。通过编程实践，我对链表的内部结构和动态内存管理有了更加深刻的理解。

任务一让我体验了头插法和尾插法建立单链表的过程。头插法由于是逆序插入，所以输出的内容会与输入的顺序相反，而尾插法则保持了输入的顺序。这两种方法的对比让我对链表的插入操作有了直观的认识。插入和删除操作的实现让我理解了链表节点之间的链接和断开是如何工作的。

任务二中，我尝试实现了插入排序算法。这个过程让我体会到了算法实现的复杂性，尤其是在调试过程中遇到的各种问题。我学会了如何通过逐步检查和修改代码来解决bug，这对于我未来解决更复杂问题是一种宝贵的经验。

任务三则是对双向循环链表的操作。与单链表相比，双向循环链表需要同时处理前驱和后继指针，这增加了操作的复杂度。我学会了如何在插入和删除操作中同时更新这两个指针，以保持链表的完整性。

总的来说，这次实验不仅加深了我对链表数据结构的理解，也锻炼了我的编程能力和问题解决能力。我认识到了理论与实践之间的差距，以及在实际编程中可能遇到的各种挑战。这些经验将对我未来的学习和工作产生积极的影响。