山东大学 软件 学院

数据结构 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201900302030 | 姓名：邵嘉明 | | 班级： 数字媒体技术 |
| 实验题目：实验三 有序链表操作 | | | |
| 实验学时：2 | | 实验日期： 2020.10.30 | |
| 实验目的：   * + 1. 掌握有序链表的基本操作：插入、删除、查找。     2. 掌握链表遍历器的使用方法。 | | | |
| 硬件环境：PC | | | |
| 软件环境：Visual Studio 2019 | | | |
| 实验步骤与内容：   1. 输入n个不为零的整数作为节点元素值，遇到0代表输入结束（不创建元素值为0的节点），创建有序链表。输出整个链表。 2. 输入一个整数，将该数插入到有有序链表相应位置。输出整个链表。 3. 输入一个整数，在链表中进行搜索，输出其在链表中的第一个出现的位置。如果不存在输出0。 4. 再一次输入一个整数，在链表中进行搜索，输出其在链表中的第一个出现的位置。如果不存在输出0。 5. 再一次输入n个不为零的整数作为节点元素值，遇到0代表输入结束（不创建元素值为0的节点），创建一个新的有序链表。输出整个链表。 6. 使用链表遍历器实现上面两个有序链表的合并，输出合并后的有序链表。 | | | |
| 结论分析与体会：  通过此次实验，我加深了对链表这一数据结构实现的理解，对链表遍历、链表排序等操作有了更深的体会。 | | | |

**本次实验主要代码：**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct chainNode{

int element;

chainNode\* previous;

chainNode\* next;

chainNode() {}

chainNode(const int& elementInput)

{

element = elementInput;

}

};

class improperIndex {};

class doubleChain{

public:

chainNode\* firstNode;

chainNode\* lastNode;

int listSize;

doubleChain(){

this->listSize = 0;

this->firstNode = this->lastNode = NULL;

}

doubleChain(chainNode\* firstNode){

this->firstNode = firstNode;

chainNode\* temp = firstNode;

this->listSize = 0;

while (temp != NULL){

temp = temp->next;

this->listSize++;

}

if (listSize == 1) lastNode = firstNode;

else lastNode = temp->previous;

}

~doubleChain(){

while (this->firstNode != NULL){

chainNode\* temp = this->firstNode->next;

delete this->firstNode;

this->firstNode = temp;

}

}

bool checkIndex(int index){

if (index < 0 || index >= this->listSize) return false;

return true;

}

void copyConstruct(int index, int element){

if (index<0 || index>this->listSize) throw improperIndex();

else if (index == 0){

chainNode\* insertElement = new chainNode(element);

insertElement->previous = NULL;

if (this->listSize != 0){

insertElement->next = this->firstNode;

this->firstNode->previous = insertElement;

}

else insertElement->next = NULL;

this->firstNode = insertElement;

chainNode\* pointer = this->firstNode;

while (pointer->next != NULL) pointer = pointer->next;

this->lastNode = pointer;

}

else if (index < this->listSize / 2) {

chainNode\* insertElement = new chainNode(element);

chainNode\* temp = this->firstNode;

for (int p = 0; p < index - 1; p++) temp = temp->next;

insertElement->previous = temp;

insertElement->next = temp->next;

temp->next->previous = insertElement;

temp->next = insertElement;

}

else if (index == this->listSize){

chainNode\* insertElement = new chainNode(element);

insertElement->next = NULL;

if (this->listSize != 0){

insertElement->previous = this->lastNode;

this->lastNode->next = insertElement;

}

else insertElement->previous = NULL;

this->lastNode = insertElement;

}

else{

chainNode\* insertElement = new chainNode(element);

chainNode\* temp = this->lastNode;

for (int p = this->listSize - 1; p > index; p--) temp = temp->previous;

insertElement->next = temp;

insertElement->previous = temp->previous;

temp->previous->next = insertElement;

temp->previous = insertElement;

}

this->listSize++;

return;

}

void find(int element) {

chainNode\* temp = this->firstNode;

int index;

for (index = 0; index < this->listSize; index++) {

if (temp->element == element) break;

temp = temp->next;

}

if (index != this->listSize) cout << index + 1 << endl;

else cout << 0 << endl;

}

void print(){

chainNode\* print = this->firstNode;

while (print != this->lastNode){

cout << print->element << ",";

print = print->next;

}

cout << this->lastNode->element << endl;

}

void orderedInsert(int element) {

chainNode\* currentNode = this->firstNode;

int index;

for (index = 0; currentNode->element < element && currentNode->next != NULL; index++) {

currentNode = currentNode->next;

}

if (currentNode->next == NULL && currentNode->element < element) this->copyConstruct(index + 1, element);

else this->copyConstruct(index, element);

}

};

int main(int argc, char\* argv[]){

vector<int> storeList1, storeList2;

int tempInput, index;

doubleChain list1, list2, list3;

cout << "Input1" << endl;

for (;;){

cin >> tempInput;

if (tempInput == 0) break;

storeList1.push\_back(tempInput);

}

sort(storeList1.begin(), storeList1.end());

for (int i = 0; i < storeList1.size(); i++) list1.copyConstruct(i, storeList1[i]);

cout << "Output1" << endl;

list1.print();

cout << "Input2" << endl;

cin >> tempInput;

list1.orderedInsert(tempInput);

storeList1.push\_back(tempInput);

cout << "Output2" << endl;

list1.print();

cout << "Input3" << endl;

cin >> tempInput;

cout << "Output3" << endl;

list1.find(tempInput);

cout << "Input4" << endl;

cin >> tempInput;

cout << "Output4" << endl;

list1.find(tempInput);

cout << "Input5" << endl;

for (;;){

cin >> tempInput;

if (tempInput == 0) break;

storeList2.push\_back(tempInput);

}

sort(storeList2.begin(), storeList2.end());

for (int i = 0; i < storeList2.size(); i++){

list2.copyConstruct(i, storeList2[i]);

storeList1.push\_back(storeList2[i]);

}

sort(storeList1.begin(), storeList1.end());

for (int i = 0; i < storeList1.size(); i++) list3.copyConstruct(i, storeList1[i]);

cout << "Output5" << endl;

list2.print();

list3.print();

cout << "End";

return 0;

}