《数据结构》实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 李爽 | 班级 | 22级软件工程专升本1班 | 学号 | 2206831522 |
| 实验名称 | **实验二单链表** | | | | |
| 实验时间 | 2023年4月13日 | | 成绩 | |  |
| 1. **实验目的：**   1. 掌握线性表的链接存储结构；  2.验证单链表及其基本操作的实现；  3.理解算法与程序的关系，能够将单链表算法转换为对应的程序。  **二、实验工具：**  软件平台：Windows7或以上版本，Visual Studio 2019  **三、实验原理：**  1、算法描述：  （1）显示函数：显示函数的作用是将单链表中存储的元素依次输出到屏幕上。具体实现过程为：从链表头结点开始，依次遍历链表中的所有结点，将每个结点中存储的元素输出到屏幕上，直到遍历完所有结点。  （2）查找函数：查找函数的作用是在单链表中查找指定元素，并返回该元素在单链表中的位置。具体实现过程为：从链表头结点开始，依次遍历链表中的所有结点，比较每个结点中存储的元素与要查找的元素是否相等，如果相等则返回该结点在链表中的位置，否则继续遍历，直到遍历完所有结点仍未找到该元素，则返回一个表示未找到的特殊值。  （3）插入函数：插入函数的作用是在单链表中插入指定元素，插入位置由用户指定。具体实现过程为：首先判断用户指定的插入位置是否合法，即是否在链表范围内。如果插入位置合法，则先创建一个新的结点，并将要插入的元素存储在该结点中。然后找到要插入位置的前一个结点，将新结点插入到该结点之后即可。  （4）删除函数：删除函数的作用是从单链表中删除指定位置的元素。具体实现过程为：首先判断用户指定的删除位置是否合法，即是否在链表范围内。如果删除位置合法，则找到要删除位置的前一个结点，将其后继结点删除即可。  （5）链表合并函数：链表合并函数的作用是将两个单链表合并成一个单链表。具体实现过程为：首先创建一个新的单链表，将第一个链表中的所有结点复制到新链表中。然后找到新链表中的最后一个结点，将第二个链表中的所有结点接到该结点之后即可。最终得到的新链表即为合并后的结果。  **四、实验步骤和内容：**  1. 建立一个单链表，随机产生10个100以内的整数，并按要求完成：  （1）编写显示函数，在屏幕上显示单链表中的10个整数；  （2）编写查找函数，从键盘输入任一整数在单链表中查找，若找到，返回该元素在单链表中的位置，否则提示无此元素；  （3）编写插入函数，从键盘输入待插入元素及插入位置，将完成插入后的单链表输出；  （4）编写删除函数，从键盘输入待删除元素位置，将该位置元素删除后的单链表输出。  2.有两个有序排序的单链表L1和L2，分别存放10个数据元素，设计算法实现两个单链表的合并，要求合并后的单链表仍然有序排序，并输出合并结果。  1.代码如下  //用于输入输出  #include <iostream>  //用于生成随机数  #include <cstdlib>  //用于获取当前时间  #include <ctime>  //命名空间  using namespace std;  // 定义单链表结点  struct Node {  //数据  int data;  //指针指向下一节点  Node\* next;  //构造函数  Node(int d) : data(d), next(nullptr) {}  };  // 定义单链表类  class LinkedList {  private:  //头结点指针  Node\* head;  //链表个数  int size;  public:  LinkedList() : head(nullptr), size(0) {}  // 随机生成10个100以内的整数并插入单链表中  void generate() {  for (int i = 0; i < 10; ++i) {  int data = rand() % 100;  insert(data, i);  }  }  // 插入元素  void insert(int data, int pos) {  //判定插入位置是否合法  if (pos < 0 || pos > size) {  cout << "无效的位置" << endl;  return;  }  //创建新节点  Node\* new\_node = new Node(data);  if (pos == 0) {  //如果插入位置为0，将新节点插入到链表头部。新节点的 "next" 指针将指向原来的头节点，  new\_node->next = head;  //然后将 "head" 指向新节点。  head = new\_node;  }  else {  //首先将指针 "curr" 指向头节点  Node\* curr = head;  //然后通过迭代从头开始移动 "pos - 1" 次，以到达插入位置之前的节点。  for (int i = 0; i < pos - 1; ++i) {  curr = curr->next;  }  //将新节点的 "next" 指针指向当前节点的 "next" 指针指向的节点  new\_node->next = curr->next;  //将当前节点的 "next" 指针指向新节点  curr->next = new\_node;  }  //链表大小+1  ++size;  }  // 删除元素  void remove(int pos) {  //判断位置是否合法  if (pos < 0 || pos >= size) {  cout << "无效的位置" << endl;  return;  }  //声明一个指向链表头部的指针 "curr"。  Node\* curr = head;  if (pos == 0) {  //将 "head" 指向原始头节点的下一个节点  head = head->next;  //删除原始头节点  delete curr;  }  else {  for (int i = 0; i < pos - 1; ++i) {  //将指针 "curr" 指向头节点，然后通过迭代从头开始移动 "pos - 1" 次，以到达要删除的节点之前的节点。  curr = curr->next;  }  //将一个指针 "temp" 指向要删除的节点  Node\* temp = curr->next;  //将当前节点的 "next" 指针指向要删除节点的下一个节点  curr->next = temp->next;  //删除节点  delete temp;  }  //链表大小-1  --size;  }  // 查找元素  int find(int data) {  //声明一个指向链表头部的指针 "curr"  Node\* curr = head;  //初始化位置 "pos" 为0  int pos = 0;  while (curr) {  if (curr->data == data) {  //如果当前节点的数据值等于要查找的数据值，则返回当前位置 "pos"  return pos;  }  //遍历查找  curr = curr->next;  ++pos;  }  //未找到  return -1;  }  // 显示单链表中的元素  void display() {  //声明一个指向链表头部的指针 "curr"，并将其初始化为头节点 "head"。  Node\* curr = head;  while (curr) {  //输出当前节点的数据值  cout << curr->data << " ";  //移动指针  curr = curr->next;  }  cout << endl;  }    // 基于归并排序的思想实现两个有序单链表的合并  LinkedList merge(LinkedList& L1, LinkedList& L2) {  // 创建新链表  LinkedList L;    // 获取头节点  Node\* curr1 = L1.head;  Node\* curr2 = L2.head;    // 遍历两个链表  while (curr1 && curr2) {  if (curr1->data <= curr2->data) {  // 如果L1中的元素更小，则将其插入新链表中  L.insert(curr1->data, L.size);  curr1 = curr1->next;  } else {  // 如果L2中的元素更小，则将其插入新链表中  L.insert(curr2->data, L.size);  curr2 = curr2->next;  }  }    // 将未被遍历完的链表中的剩余元素插入新链表中  while (curr1) {  L.insert(curr1->data, L.size);  curr1 = curr1->next;  }  while (curr2) {  L.insert(curr2->data, L.size);  curr2 = curr2->next;  }    return L;  }  };  int main() {  srand(time(NULL));  LinkedList list;  list.generate();  cout << "生成的链表为：";  list.display();  int data, pos;  cout << "请输入要查找的整数：";  cin >> data;  pos = list.find(data);  if (pos != -1) {  cout << data << "的位置为" << pos << endl;  }  else {  cout << "该元素不存在" << endl;  }  cout << "请输入要插入的整数和位置：";  cin >> data >> pos;  list.insert(data, pos);  cout << "插入元素后的链表为：";  list.display();  cout << "请输入要删除的位置：";  cin >> pos;  list.remove(pos);  cout << "删除元素后的链表为：";  list.display();  return 0;  }    2. 代码如下  **我们稍微修改一下主函数的调用即可，实现函数在第一题中已经给出**  int main() {  // 设置随机数种子  srand(time(nullptr));  // 创建两个有序单链表  LinkedList L1, L2;  // 生成随机数据并插入链表中  L1.generate();  L2.generate();  // 显示原始单链表  cout << "L1: ";  L1.display();  cout << "L2: ";  L2.display();  // 合并单链表并显示结果  LinkedList L3 = L1.merge(L1,L2);  cout << "合并后的单链表: ";  L3.display();  return 0;  }    **五、实验总结：**  本次实验中，我使用C++语言完成了单链表结构的创建，并加入了两个链表的合并函数。在实验过程中，我成功地实现了基本的增删改查功能，并通过调用主函数完成了测试。  在实验中，我首先学习了单链表的基本概念和实现方式，包括单链表节点的定义、插入、删除、查找等操作。在此基础上，我进一步学习了链表的合并算法，使用并归算法完成函数。  在实现过程中，我遇到了一些问题，例如在链表的删除操作中，需要考虑头节点和尾节点的特殊情况；在链表的合并算法中，需要特别注意空链表和链表长度的情况。通过认真思考和调试，我最终成功地解决了这些问题，并完成了实验的所有要求。  通过这次实验，我深入了解了单链表的基本概念和实现方式，加深了对C++语言的理解和应用，提高了编程能力和实践能力。同时，我也学会了如何认真思考问题、解决问题，锻炼了自己的实验能力和创新能力。  **六、教师评语：** | | | | | |