《数据结构》实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 马昆 | 班级 | 软件工程专升本1班 | 学号 | 2206831544 |
| 实验名称 | **实验二单链表** | | | | |
| 实验时间 | 2023年4月10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：在单链表类中添加一个PrintList()成员方法，如果链表为空就直接打印{}，如果链表不为空就先定义一个中间变量，用于存放链表遍历的当前结点，第一个节点设置为第一个元素节点，定义while循环，条件是如果当前节点的下一个节点不为空就继续循环，循环体内就打印当前节点的数据然后将当前节点的指针变为当前节点的下一节点的指针，最终打印效果{11,22,33,44}。  （2）查找函数：按位查找，如果查找的位置不在1~数据总个数的范围，就直接抛异常。如果在范围内，就定义一个中间变量，表示当前遍历的数据节点，并第一个当前节点为头节点。定义一个for循环，从0开始，到要查找的下标-1结束，每循环一次就将当前节点指针赋值为当前节点的下一节点的指针。最后循环结束返回当前节点的数据。  （3）插入函数：在指定位置插入元素，如果要插入的位置不在1~链表长度-1的差的范围就抛出异常。如果在范围内，就创建一个中间节点指针用于指向当前遍历到的节点，当前节点的第一个节点设置为头节点，并且为新数据创建一个新的节点。建立一个for循环，循环插入位置-1次，每次循环都将当前节点指针变为当前节点的下一节点的指针。循环结束后将新节点的下一节点指针设置为当前节点的下一节点指针，当前节点的下一节点指针设置为新节点的指针。  （4）删除函数：删除指定位置的数据，如果位置不在1~链表长度的范围就抛出异常。如果在范围内就先使用查找函数找到对应下表的数据，暂时存放在一个变量中，方便最后返回。创建一个遍历的当前节点指针变量，第一个当前节点设置为头节点，再创建一个临时节点指针用于存放最后需要删除并销毁的节点，初始化为空指针。定义一个for循环，循环需要删除位置的次数，每次循环都将当前节点指针变为当前节点的下一节点的指针。循环结束后将临时变量赋值为当前节点的指针，然后将当前节点的指针改为当前节点的下一节点的指针。最后回收临时节点指针所指向的空间，然后返回预先存放的要删除的数据。  （5）链表合并函数：先创建一个新的空单链表，然后使用两个for循环分别插入两个链表中的数据，然后返回新的单链表的指针。  **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；   |  | | --- | | // 遍历操作，按序号依次输出各结点  template <typename DataType>  void LinkList<DataType>::PrintList() {  if (this->Empty()) {  cout << "{}" << endl;  return;  }  Node<DataType>\* temp = header->next;  cout << "{" << temp->data;  while (temp->next != nullptr) {  temp = temp->next;  cout << "," << temp->data;  }  cout << "}" << endl;  } | | #include <iostream>  #include "LinkList.h"  #include <random>  using namespace std;  // 菜单  template <typename DataType = int>  void Menu(int choose, LinkList<>\* list);  int main() {  default\_random\_engine rand; // 创建随机数引擎  rand.seed(time(0)); // 将当前时间的时间戳作为种子  int init[10]{}; // 定义用于装10个随机1~100的数字的数组容器  // 随机10个1~100的数字  for (int i = 0; i < 10; i++) {  init[i] = rand() % 101;  }  LinkList<> list(init, 10); // 将数字装进链表中  list.PrintList();  } | |  |   （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；   |  | | --- | | // 按值查找，查找值为x的结点  // 0：没找到  template <typename DataType>  int LinkList<DataType>::Locate(DataType x) {  Node<DataType>\* temp = header;  for (int i = 0; i < this->Length(); i++) {  temp = temp->next;  if (temp->data == x) {  return i + 1;  }  }  return 0;  } | | #include <iostream>  #include "LinkList.h"  #include <random>  using namespace std;  // 菜单  template <typename DataType = int>  void Menu(int choose, LinkList<>\* list);  int main() {  default\_random\_engine rand; // 创建随机数引擎  rand.seed(time(0)); // 将当前时间的时间戳作为种子  int init[10]{}; // 定义用于装10个随机1~100的数字的数组容器  // 随机10个1~100的数字  for (int i = 0; i < 10; i++) {  init[i] = rand() % 101;  }  LinkList<> list(init, 10); // 将数字装进链表中  list.PrintList();  cout << "请输入你要查找的数字：";  int num = 0;  cin >> num;  if (num = list.Locate(num)) {  cout << "位置为" << num;  }  else {  cout << "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；   |  | | --- | | // 插入操作，在第i个位置插入值为x的结点  template <typename DataType>  void LinkList<DataType>::Insert(int i, DataType x) {  if (i < 1 || i > this->Length() + 1) throw "位置错误";  Node<DataType>\* temp = header, \* new\_node = new Node<DataType>;  for (int j = 0; j < i - 1; j++) {  temp = temp->next;  }  new\_node->data = x;  new\_node->next = temp->next;  temp->next = new\_node;  } | | #include <iostream>  #include "LinkList.h"  #include <random>  using namespace std;  // 菜单  template <typename DataType = int>  void Menu(int choose, LinkList<>\* list);  int main() {  default\_random\_engine rand; // 创建随机数引擎  rand.seed(time(0)); // 将当前时间的时间戳作为种子  int init[10]{}; // 定义用于装10个随机1~100的数字的数组容器  // 随机10个1~100的数字  for (int i = 0; i < 10; i++) {  init[i] = rand() % 101;  }  LinkList<> list(init, 10); // 将数字装进链表中  list.PrintList();  cout << "请输入你要插入的位置与插入的数字：";  int index = 0,num = 0;  cin >> index >> num;    list.Insert(index, num);    cout << "插入后结果：" << endl;  list.PrintList();  } | |  |   （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。   |  | | --- | | // 删除操作，删除第i个结点  template <typename DataType>  DataType LinkList<DataType>::Delete(int i) {  if (i < 1 || i > this->Length()) throw "位置错误";  DataType del = this->Get(i);  Node<DataType>\* node = header, \* temp = nullptr;  for (int j = 0; j < i-1; j++) {  node = node->next;  }  temp = node->next;  node->next = node->next->next;  delete(temp);  return del;  } | | #include <iostream>  #include "LinkList.h"  #include <random>  using namespace std;  // 菜单  template <typename DataType = int>  void Menu(int choose, LinkList<>\* list);  int main() {  default\_random\_engine rand; // 创建随机数引擎  rand.seed(time(0)); // 将当前时间的时间戳作为种子  int init[10]{}; // 定义用于装10个随机1~100的数字的数组容器  // 随机10个1~100的数字  for (int i = 0; i < 10; i++) {  init[i] = rand() % 101;  }  LinkList<> list(init, 10); // 将数字装进链表中  list.PrintList();  cout << "请输入你要删除的元素的位置：";  int index = 0;  cin >> index;  cout << "你删除的数字是：" << list.Delete(index) << endl;    cout << "删除后结果：" << endl;  list.PrintList();  } | |  |   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。   |  | | --- | | // 将两个链表合并为一个链表  template <typename DataType>  LinkList<DataType>\* LinkList<DataType>::Merge(LinkList<DataType>\* first, LinkList<DataType>\* second) {  LinkList<DataType> list;  for (int i = 1; i <= second->Length(); i++) {  list.Insert(1, second->Get(i));  }  for (int i = 1; i <= first->Length(); i++) {  list.Insert(1,first->Get(i));  }  return &list;  } | | #include <iostream>  #include "LinkList.h"  using namespace std;  // 菜单  template <typename DataType = int>  void Menu(int choose, LinkList<>\* list);  int main() {  int init1[] = { 1,2,3,4,5,6,7,8,9,10 };  int init2[] = { 11,12,13,14,15,16,17,18,19,20 };  LinkList<> list1(init1, 10);  LinkList<> list2(init2, 10);  LinkList<>\* list3 = list1.Merge(&list1, &list2);  list3->PrintList();  } | |  |   **五、实验总结：**  **在本次实验中，其中的过程还是比较曲折的，由于自己的粗心引发了很多不应该出现的问题，因为本次实验与之前的作业1很相似，然后复制了作业1的代码，但是忘记了当初实现的时候接口下标是从1开始的，然后自己就一直报错非法的内存访问，因为自己嫌麻烦就一直没有打断点去看，链表合并的实现就换了一个又一个，最后还是用最笨的方法实现了，这次实验让我明白了，使用封装好的代码一定要去看一看注释，我连我自己的注释都没看。**  **六、教师评语：** | | | | | |