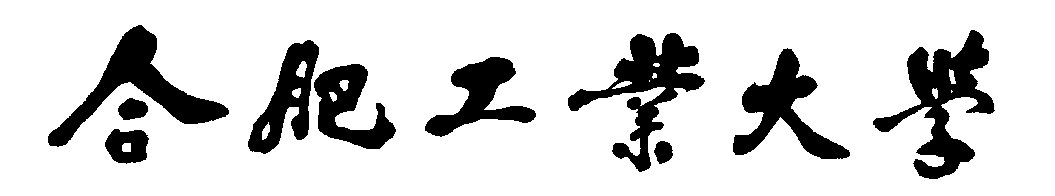
****

计算机与信息学院

数据结构实验报告

|  |  |
| --- | --- |
| 专 业 班 级 | 物联网一班 |
| 学生姓名及学号 | 敬成超 2023212388 |
| 课程教学班号 |  |
| 任 课 教 师 | 胡学钢 |
| 实验指导教师 |  |
| 实验地点 | C栋304 |
| 2023 ~2024 学年第二学期 | |

实验序号及名称：实验 二 **循环链表、双链表及链表应用实验**

实验时间∶ 3 月 21日

|  |
| --- |
| 预习内容 |
| 一、实验目的和要求∶  (1)理解单循环链表及双循环链表的特点。  (2)掌握这两种结构的算法设计。  (3)运用链表存储数据设计有关算法。  (4)理解头结点、头指针概念以及设置头结点的优点 |
| 二、实验任务∶  **要求：在实现单循环链表和双链表的类基础上，完成下列内容**：  **算法设计**  **<1>设计算法依次访问无头结点的单循环链表的各结点。**  实验测试数据基本要求：  第一组数据：链表元素为 （1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60）  第二组数据：链表元素为 （10, 20, 30, 40, 50, 60, 70, 80, 90, 100）  **<2>设计算法以判断一个带头结点的单循环链表是否满足这样的条件：**  **其中每个结点的元素值与其序号的差的绝对值不大于３。**  若成立, 返回TRUE, 否则返回FALSE。  实验测试数据基本要求：  第一组数据：链表元素为  （1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18）  第二组数据：链表元素为  （1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 20, 18）  **<3>利用递增有序的单循环链表表示集合, 分别求两个链表表示的集合的交、并集所构成的链表。**  实验测试数据基本要求：  第一组  第一个链表元素为 （1, 3, 6, 10, 15, 16, 17, 18, 19, 20）  第二个链表元素为 （1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 18, 20, 30）  第二组  第一个链表元素为 （1, 3, 6, 10, 15, 16, 17, 18, 19, 20）  第二个链表元素为 （2, 4, 5, 7, 8, 9, 12, 22）  第三组  第一个链表元素为 （）  第二个链表元素为 （1, 2, 3, 4, 5, 6, 7, 8, 9, 10）  **<4>设计算法以构造带头结点的双循环链表。**  实验测试数据基本要求：  第一组数据：链表元素为 （1, 2, 3, 4, 5, 6, 7, 8, 9, 10）  第二组数据：链表元素为 （10, 30, 40, 55, 60, 70, 88, 99, 100）  **<5>编写算法以判断一个带头结点的双循环链表是否是对称的, 若成立, 返回TRUE, 否则返回FALSE。**  实验测试数据基本要求：  第一组数据：链表元素为 （1, 2, 3, 4, 5, 4, 3, 2, 1）  第二组数据：链表元素为 （1, 2, 3, 4, 5, 5, 4, 3, 2, 1）  第三组数据：链表元素为 （1, 2, 3, 4, 5, 6, 3, 2, 1）  第四组数据：链表元素为 （1, 2, 3, 4, 5, 5, 6, 4, 3, 2, 1） |
| 三、实验准备方案，包括以下内容：  （硬件类实验：实验原理、实验线路、设计方案等）  （软件类实验：所采用的核心方法、框架或流程图及程序清单） |

|  |
| --- |
| 实验内容 |
| 一、实验用仪器、设备：  电脑 |
| 二、实验内容与步骤（过程及数据记录）：  **实验1: 访问无头结点的单循环链表的各结点**   1. **初始化链表**：使用给定的第一组或第二组数据创建单循环链表。 2. **找到入口点**：由于无头结点，首先需要遍历链表找到入口点，即第一个非空节点。 3. **访问节点**：从入口点开始，使用一个指针按顺序访问每个节点，直到再次回到入口点。 4. **输出节点值**：在访问过程中，打印每个节点的值。 5. **测试**：使用第一组和第二组数据测试算法，确保所有节点都被正确访问。   **实验2: 判断单循环链表是否满足特定条件**   1. **初始化链表**：使用给定的第一组或第二组数据创建带头结点的单循环链表。 2. **遍历链表**：从头结点开始，按顺序访问每个数据节点。 3. **条件检查**：对于每个节点，计算节点值与序号的差的绝对值，检查是否不大于3。 4. **结果判断**：如果所有节点都满足条件，则返回TRUE；否则，在发现第一个不满足条件的节点时返回FALSE。 5. **测试**：使用第一组和第二组数据测试算法，验证结果的正确性。   **实验3: 求两个链表表示的集合的交、并集**   1. **初始化链表**：创建两个单循环链表，分别用第一组和第二组数据。 2. **求交集**：遍历第一个链表，对于每个节点，检查其值是否存在于第二个链表中，如果存在，则将该值添加到新的链表中。 3. **求并集**：创建一个新链表，将两个链表中的所有不重复元素按递增顺序添加到新链表中。 4. **输出结果**：打印交集和并集链表的元素。 5. **测试**：确保算法能够正确处理不同的输入数据。   **实验4: 构造带头结点的双循环链表**   1. **设计节点结构**：设计一个节点类，包含数据域和两个指针，分别指向前一个和后一个节点。 2. **初始化链表**：创建一个带头结点的双循环链表，头结点可以为空。 3. **添加节点**：实现一个函数，将新节点添加到链表的末尾，并更新前一个节点和后一个节点的指针。 4. **遍历链表**：实现一个函数，能够从任意节点开始，向前和向后遍历整个链表。 5. **测试**：使用第一组和第二组数据测试算法，确保链表正确构造和遍历。   **实验5: 判断双循环链表是否对称**   1. **初始化链表**：使用给定的某组数据创建带头结点的双循环链表。 2. **找到中点**：向前遍历链表，找到链表的中点。 3. **从两端开始比较**：从链表的头部和中点开始，分别向前和向后遍历，比较对应的节点值是否相等。 4. **结果判断**：如果所有对应的节点值都相等，则链表是对称的，返回TRUE；否则，返回FALSE。 5. **测试**：使用所有给定的数据组测试算法，验证对称性的判断是否正确。   #include <iostream>  #include <vector>  // 定义链表节点结构  struct Node {  int data;  Node\* next;  Node(int val) : data(val), next(nullptr) {}  };  // 创建单循环链表  Node\* createCircularLinkedList(const std::vector<int>& elements) {  if (elements.empty()) return nullptr;  Node\* head = new Node(elements[0]);  Node\* current = head;  for (size\_t i = 1; i < elements.size(); ++i) {  current->next = new Node(elements[i]);  current = current->next;  }  // 将最后一个节点的next指向head，形成循环  current->next = head;  return head;  }  // 依次访问单循环链表的各结点  void traverseCircularLinkedList(Node\* head) {  if (!head) return;  Node\* current = head;  do {  std::cout << current->data << " ";  current = current->next;  } while (current != head);  std::cout << std::endl;  }  // 释放链表内存  void deleteCircularLinkedList(Node\* head) {  if (!head) return;  Node\* current = head;  Node\* next = nullptr;  do {  next = current->next;  delete current;  current = next;  } while (current != head);  }  int main() {  // 第一组数据  std::vector<int> elements1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60 };  Node\* head1 = createCircularLinkedList(elements1);  std::cout << "第一组数据：";  traverseCircularLinkedList(head1);  deleteCircularLinkedList(head1);  // 第二组数据  std::vector<int> elements2 = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };  Node\* head2 = createCircularLinkedList(elements2);  std::cout << "第二组数据：";  traverseCircularLinkedList(head2);  deleteCircularLinkedList(head2);  return 0;  }  #include <iostream>  #include <vector>  #include <cmath>  // 定义链表节点结构  struct Node {  int data;  Node\* next;  Node(int val) : data(val), next(nullptr) {}  };  // 创建带头结点的单循环链表  Node\* createCircularLinkedList(const std::vector<int>& elements) {  if (elements.empty()) return nullptr;  Node\* head = new Node(0); // 头结点  Node\* current = head;  for (size\_t i = 0; i < elements.size(); ++i) {  current->next = new Node(elements[i]);  current = current->next;  }  // 将最后一个节点的next指向head->next，形成循环  current->next = head->next;  return head;  }  // 判断链表是否满足条件  bool checkCondition(Node\* head) {  if (!head || !head->next) return true; // 空链表或只有一个头结点  Node\* current = head->next;  int index = 1;  do {  if (std::abs(current->data - index) > 3) {  return false;  }  current = current->next;  ++index;  } while (current != head->next);  return true;  }  // 释放链表内存  void deleteCircularLinkedList(Node\* head) {  if (!head) return;  Node\* current = head->next;  Node\* next = nullptr;  do {  next = current->next;  delete current;  current = next;  } while (current != head->next);  delete head; // 删除头结点  }  int main() {  // 第一组数据  std::vector<int> elements1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 18 };  Node\* head1 = createCircularLinkedList(elements1);  std::cout << "第一组数据：" << (checkCondition(head1) ? "TRUE" : "FALSE") << std::endl;  deleteCircularLinkedList(head1);  // 第二组数据  std::vector<int> elements2 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 20, 18 };  Node\* head2 = createCircularLinkedList(elements2);  std::cout << "第二组数据：" << (checkCondition(head2) ? "TRUE" : "FALSE") << std::endl;  deleteCircularLinkedList(head2);  return 0;  }  #include <iostream>  #include <vector>  // 定义链表节点结构  struct Node {  int data;  Node\* next;  Node(int val) : data(val), next(nullptr) {}  };  // 创建单循环链表  Node\* createCircularLinkedList(const std::vector<int>& elements) {  if (elements.empty()) return nullptr;  Node\* head = new Node(elements[0]);  Node\* current = head;  for (size\_t i = 1; i < elements.size(); ++i) {  current->next = new Node(elements[i]);  current = current->next;  }  // 将最后一个节点的next指向head，形成循环  current->next = head;  return head;  }  // 求两个递增有序单循环链表的交集  Node\* intersection(Node\* head1, Node\* head2) {  if (!head1 || !head2) return nullptr;  Node\* resultHead = nullptr;  Node\* resultTail = nullptr;  Node\* current1 = head1;  Node\* current2 = head2;  do {  if (current1->data == current2->data) {  if (!resultHead) {  resultHead = new Node(current1->data);  resultTail = resultHead;  }  else {  if (resultTail->data != current1->data) {  resultTail->next = new Node(current1->data);  resultTail = resultTail->next;  }  }  current1 = current1->next;  current2 = current2->next;  }  else if (current1->data < current2->data) {  current1 = current1->next;  }  else {  current2 = current2->next;  }  } while (current1 != head1 && current2 != head2);  if (resultHead) {  resultTail->next = resultHead;  }  return resultHead;  }  // 求两个递增有序单循环链表的并集  Node\* unionSet(Node\* head1, Node\* head2) {  if (!head1) return head2;  if (!head2) return head1;  Node\* resultHead = nullptr;  Node\* resultTail = nullptr;  Node\* current1 = head1;  Node\* current2 = head2;  do {  if (current1->data == current2->data) {  if (!resultHead) {  resultHead = new Node(current1->data);  resultTail = resultHead;  }  else {  if (resultTail->data != current1->data) {  resultTail->next = new Node(current1->data);  resultTail = resultTail->next;  }  }  current1 = current1->next;  current2 = current2->next;  }  else if (current1->data < current2->data) {  if (!resultHead) {  resultHead = new Node(current1->data);  resultTail = resultHead;  }  else {  if (resultTail->data != current1->data) {  resultTail->next = new Node(current1->data);  resultTail = resultTail->next;  }  }  current1 = current1->next;  }  else {  if (!resultHead) {  resultHead = new Node(current2->data);  resultTail = resultHead;  }  else {  if (resultTail->data != current2->data) {  resultTail->next = new Node(current2->data);  resultTail = resultTail->next;  }  }  current2 = current2->next;  }  } while (current1 != head1 && current2 != head2);  while (current1 != head1) {  if (resultTail->data != current1->data) {  resultTail->next = new Node(current1->data);  resultTail = resultTail->next;  }  current1 = current1->next;  }  while (current2 != head2) {  if (resultTail->data != current2->data) {  resultTail->next = new Node(current2->data);  resultTail = resultTail->next;  }  current2 = current2->next;  }  if (resultHead) {  resultTail->next = resultHead;  }  return resultHead;  }  // 打印单循环链表  void printCircularLinkedList(Node\* head) {  if (!head) return;  Node\* current = head;  do {  std::cout << current->data << " ";  current = current->next;  } while (current != head);  std::cout << std::endl;  }  // 释放链表内存  void deleteCircularLinkedList(Node\* head) {  if (!head) return;  Node\* current = head;  Node\* next = nullptr;  do {  next = current->next;  delete current;  current = next;  } while (current != head);  }  int main() {  // 第一组数据  std::vector<int> elements1\_1 = { 1, 3, 6, 10, 15, 16, 17, 18, 19, 20 };  std::vector<int> elements1\_2 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 18, 20, 30 };  Node\* head1\_1 = createCircularLinkedList(elements1\_1);  Node\* head1\_2 = createCircularLinkedList(elements1\_2);  Node\* intersection1 = intersection(head1\_1, head1\_2);  Node\* union1 = unionSet(head1\_1, head1\_2);  std::cout << "第一组交集：";  printCircularLinkedList(intersection1);  std::cout << "第一组并集：";  printCircularLinkedList(union1);  deleteCircularLinkedList(head1\_1);  deleteCircularLinkedList(head1\_2);  deleteCircularLinkedList(intersection1);  deleteCircularLinkedList(union1);  // 第二组数据  std::vector<int> elements2\_1 = { 1, 3, 6, 10, 15, 16, 17, 18, 19, 20 };  std::vector<int> elements2\_2 = { 2, 4, 5, 7, 8, 9, 12, 22 };  Node\* head2\_1 = createCircularLinkedList(elements2\_1);  Node\* head2\_2 = createCircularLinkedList(elements2\_2);  Node\* intersection2 = intersection(head2\_1, head2\_2);  Node\* union2 = unionSet(head2\_1, head2\_2);  std::cout << "第二组交集：";  printCircularLinkedList(intersection2);  std::cout << "第二组并集：";  printCircularLinkedList(union2);  deleteCircularLinkedList(head2\_1);  deleteCircularLinkedList(head2\_2);  deleteCircularLinkedList(intersection2);  deleteCircularLinkedList(union2);  // 第三组数据  std::vector<int> elements3\_1 = {};  std::vector<int> elements3\_2 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  Node\* head3\_1 = createCircularLinkedList(elements3\_1);  Node\* head3\_2 = createCircularLinkedList(elements3\_2);  Node\* intersection3 = intersection(head3\_1, head3\_2);  Node\* union3 = unionSet(head3\_1, head3\_2);  std::cout << "第三组交集：";  printCircularLinkedList(intersection3);  std::cout << "第三组并集：";  printCircularLinkedList(union3);  deleteCircularLinkedList(head3\_1);  deleteCircularLinkedList(head3\_2);  deleteCircularLinkedList(intersection3);  deleteCircularLinkedList(union3);  return 0;  }  #include <iostream>  #include <vector>  // 定义双循环链表节点结构  struct Node {  int data;  Node\* prev;  Node\* next;  Node(int val) : data(val), prev(nullptr), next(nullptr) {}  };  // 创建带头结点的双循环链表  Node\* createDoublyCircularLinkedList(const std::vector<int>& elements) {  if (elements.empty()) return nullptr;  // 创建头结点  Node\* head = new Node(0);  Node\* current = head;  // 创建链表节点  for (int element : elements) {  Node\* newNode = new Node(element);  newNode->prev = current;  current->next = newNode;  current = newNode;  }  // 将最后一个节点的next指向头结点的next，形成循环  current->next = head->next;  // 将头结点的next的prev指向最后一个节点，形成循环  head->next->prev = current;  return head;  }  // 打印双循环链表  void printDoublyCircularLinkedList(Node\* head) {  if (!head || !head->next) return;  Node\* current = head->next;  do {  std::cout << current->data << " ";  current = current->next;  } while (current != head->next);  std::cout << std::endl;  }  // 释放双循环链表内存  void deleteDoublyCircularLinkedList(Node\* head) {  if (!head || !head->next) return;  Node\* current = head->next;  Node\* next = nullptr;  do {  next = current->next;  delete current;  current = next;  } while (current != head->next);  delete head; // 删除头结点  }  int main() {  // 第一组数据  std::vector<int> elements1 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  Node\* head1 = createDoublyCircularLinkedList(elements1);  std::cout << "第一组数据：";  printDoublyCircularLinkedList(head1);  deleteDoublyCircularLinkedList(head1);  // 第二组数据  std::vector<int> elements2 = { 10, 30, 40, 55, 60, 70, 88, 99, 100 };  Node\* head2 = createDoublyCircularLinkedList(elements2);  std::cout << "第二组数据：";  printDoublyCircularLinkedList(head2);  deleteDoublyCircularLinkedList(head2);  return 0;  }  #include <iostream>  struct Node {  int data;  Node\* next;  Node\* prev;  };  bool isSymmetric(Node\* head) {  if (head->next == head) {  // 空链表或只有一个元素的链表是对称的  return true;  }  Node\* front = head->next;  Node\* back = head->prev;  while (front != back && front->prev != back) {  if (front->data != back->data) {  return false;  }  front = front->next;  back = back->prev;  }  return true;  }  // 辅助函数：创建双循环链表  Node\* createDoublyCircularLinkedList(int arr[], int n) {  Node\* head = new Node();  head->next = head;  head->prev = head;  Node\* current = head;  for (int i = 0; i < n; ++i) {  Node\* newNode = new Node();  newNode->data = arr[i];  newNode->next = head;  newNode->prev = current;  current->next = newNode;  head->prev = newNode;  current = newNode;  }  return head;  }  // 辅助函数：打印链表（用于调试）  void printList(Node\* head) {  Node\* current = head->next;  while (current != head) {  std::cout << current->data << " ";  current = current->next;  }  std::cout << std::endl;  }  int main() {  int arr1[] = { 1, 2, 3, 4, 5, 4, 3, 2, 1 };  int arr2[] = { 1, 2, 3, 4, 5, 5, 4, 3, 2, 1 };  int arr3[] = { 1, 2, 3, 4, 5, 6, 3, 2, 1 };  int arr4[] = { 1, 2, 3, 4, 5, 5, 6, 4, 3, 2, 1 };  Node\* head1 = createDoublyCircularLinkedList(arr1, sizeof(arr1) / sizeof(arr1[0]));  Node\* head2 = createDoublyCircularLinkedList(arr2, sizeof(arr2) / sizeof(arr2[0]));  Node\* head3 = createDoublyCircularLinkedList(arr3, sizeof(arr3) / sizeof(arr3[0]));  Node\* head4 = createDoublyCircularLinkedList(arr4, sizeof(arr4) / sizeof(arr4[0]));  std::cout << "第一组数据: " << (isSymmetric(head1) ? "TRUE" : "FALSE") << std::endl;  std::cout << "第二组数据: " << (isSymmetric(head2) ? "TRUE" : "FALSE") << std::endl;  std::cout << "第三组数据: " << (isSymmetric(head3) ? "TRUE" : "FALSE") << std::endl;  std::cout << "第四组数据: " << (isSymmetric(head4) ? "TRUE" : "FALSE") << std::endl;  return 0;  } |

|  |
| --- |
|  |
| 三、实验结果分析、思考题解答∶  实验结果符合预期，没有思考题 |
| 四、感想、体会、建议∶  感想   1. **理论与实践结合**：通过实际编写代码和测试，我深刻体会到理论知识与实际操作之间的联系。书本上的知识在实际编程中得到了应用和验证。 2. **问题解决能力的提升**：在实验过程中，我遇到了一些问题，如指针操作的复杂性、边界条件的处理等。通过查找资料和调试代码，我逐渐解决了这些问题，这锻炼了我的问题解决能力。 3. **对数据结构的理解加深**：通过亲手实现双循环链表和对称性判断，我对双链表的结构和操作有了更深入的理解。   体会   1. **编程细节的重要性**：在编写代码时，每一个细节都可能导致程序出错。例如，指针的初始化和链表的循环条件设置都需要非常小心。 2. **调试技巧的提升**：通过这次实验，我学会了如何使用调试工具来逐步检查代码，这对我理解程序运行过程和发现错误非常有帮助。 3. **算法优化的意识**：在实现对称性判断时，我意识到算法的时间复杂度和空间复杂度的重要性，这激发了我对算法优化的兴趣。   建议   1. **增加更多实验案例**：建议老师或教材提供更多不同类型的双链表操作实验，以增加我们对不同情况的处理能力。 2. **小组讨论与分享**：建议组织小组讨论，让同学们分享各自的编程经验和遇到的问题，这样可以互相学习，共同进步。 3. **提供详细的实验指导**：对于初学者，提供详细的实验指导和常见问题解答，可以帮助我们更快地掌握实验要点，减少不必要的困惑。 4. **结合实际应用**：建议实验内容结合实际应用场景，这样可以使我们更好地理解数据结构在实际问题中的应用价值。   通过这次实验，我不仅学到了双链表的相关知识，还提升了自己的编程能力和解决问题的能力。我相信这些经验和技能将对我的未来学习和工作产生积极影响。 |
| 实验成绩∶  指导教师签名：  年 月 日 |