**数据结构实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号-姓名 |  | 实验时间 | 2023年 9月 27日 |
| 诚信声明 | 手写，签名 | | |
| 实验题目 | 一元多项式相加、减运算器 | | |
| 实验过程中遇到的主要问题 | 一元稀疏多项式的存储方式：要实现一元多项式的加减，要选择合适的数据结构来表示多项式。若使用顺序表存储，只需要保存系数，但对于稀疏多项式的存储效率低下。为了同时实现项数的存储与稀疏多项式的高效率存储，可以使用带头节点的单链表，用头节点存储项数，每个节点存储系数和指数。为了输出方便，按指数降序排列。  多项式相加时的插入：实现多项式相加的功能，为了保持新多项式有序，需要在合适的位置进行插入。需要实现简单搜索算法来找到插入的位置。 | | |
| 实验小结 | 本次实验实现了一元多项式的加减计算器，包括多项式输入、输出以及相加、相减等功能。通过使用带头结点的单向链表，并按照指数降序来存储多项式，能够方便地表示多项式的项数和每一项的系数与指数。在多项式相加和相减的算法实现过程中，需要注意指数相同时的处理以及如何正确插入新的项。 | | |
| 数据结构  （自定义数据类型） | 数据结构：一元多项式  typedef struct Term {  int coef, exp; // 分别是系数和指数  struct Term \*next; // 指向下一项的指针  } Term, \*Poly; // 一元多项式的链表存储 | | |
| 主要算法  （或算法说明） | 1. 输入多项式并建立链表 2. 创建一个空的多项式链表。 3. 读入多项式项数n，置循环计数器i为0 4. 读入多项式的系数和指数，并创建多项式节点。 5. 读入的多项式是指数升序，将创建的节点用头插法插入到链表中，按指数降序排列。 6. 更新头结点的项数。 7. 如果i<n，返回②，否则退出 8. 输出多项式 9. 输出头结点的项数。 10. 遍历多项式链表，依次输出每个节点的系数和指数。 11. 多项式相加 12. 创建一个空的结果链表。 13. 使用两个指针分别指向两个多项式链表的头结点。 14. 循环比较指针指向的节点的指数大小： 15. 如果指数相同，将两个节点的系数相加，创建一个新的节点存储结果，并将新节点插入到结果链表中，并移动两个指针。如果指数不同，将指数较大的节点插入到结果链表中，并移动对应的指针。 16. 遍历完其中一个链表后，将另一个链表中剩余的节点插入到结果链表中。 17. 返回结果链表。 18. 多项式相减 19. 创建一个空的结果链表。 20. 使用两个指针分别指向两个多项式链表的头结点。 21. 循环比较指针指向的节点的指数大小： 22. 如果指数相同，将两个节点的系数相减，创建一个新的节点存储相减结果，并将新节点插入到结果链表中，并移动两个指针。如果指数不同，将指数较大的节点插入到结果链表中，但需要将作为减数的多项式的剩余节点的系数取相反数，最后移动对应的指针。 23. 遍历完其中一个链表后，将另一个链表中剩余的节点插入到结果链表中，但需要将作为减数的多项式的剩余节点的系数取相反数。 24. 返回结果链表。 | | |

**数据结构实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号-姓名 |  | 实验时间 | 2023年 9月 27日 |
| 诚信声明 | 手写，签名 | | |
| 实验题目 | 单链表上的排序运算 | | |
| 实验过程中遇到的主要问题 | 冒泡时的元素交换：由于单链表的结构特点，节点之间的交换不是直接交换节点的值，而是需要调整节点之间的指针连接关系。  当需要交换相邻节点的位置时，需要同时修改两对节点之间的指针连接，以保持链表的完整性。假设有几个相邻节点 A、B、C和 D，如果要将节点B移动到节点C的位置，就需要先保存B、C，将节点A的next指针指向节点C，将节点B的next指针指向节点C的next指针，再将节点C的next指针指向B。这意味着需要在遍历链表时记录上一个节点的位置，以便在交换节点后正确连接链表。 | | |
| 实验小结 | 本次实验实现了单链表上的排序运算，主要涉及单链表的创建、遍历和排序操作。通过实验，我了解了单链表的基本操作和单链表上排序算法的实现。在实现过程中，需要熟悉单链表的节点结构和指针操作，并根据单链表的特点进行相应的实现。 | | |
| 数据结构  （自定义数据类型） | 数据结构：单链表节点结构  struct Node {  int data; // 节点存储的值  Node \*next; // 指向下一个节点的指针  } | | |
| 主要算法  （或算法说明） | 1. 创建单链表：   通过循环读入数据，并创建新的节点，使用尾插法将节点连接到链表上，直到输入结束。   1. 遍历单链表并输出元素序列：   通过循环遍历链表的每个节点，从头节点开始，直到链表尾部（即节点的next指针为空）。   1. 冒泡排序： 2. 初始化指针和标志：定义一个当前节点指针p，一个上一次处理的最后节点指针last，以及一个交换标志swapped，初始时将swapped设置为1。 3. 进入排序循环：使用do-while循环，在每次循环开始时将swapped设置为 0。 4. 遍历链表并比较相邻节点：在内部循环中，从头节点开始通过链表，对每对相邻节点进行比较。 5. 执行元素交换：如果当前节点的值大于其后继节点的值，需要进行元素交换。这时，保留后继节点指针tmp，更改前一个节点pre的next指针，将其指向后继节点tmp，并将当前节点p的next指针指向后继节点tmp的后继节点，最后将后继节点tmp的next指针指向当前节点p。 6. 更新标志：在进行了元素交换后，将swapped设置为1，表示发生了交换，以便下一轮循环继续执行。 7. 更新上一次处理的最后节点指针：在完成一次内部循环后，将last更新为当前节点p。 8. 继续循环直到排序完成：继续执行外部循环，直到swapped的值为0，表示没有发生交换，即链表已经排好序。 9. 完成排序：排序完成 | | |