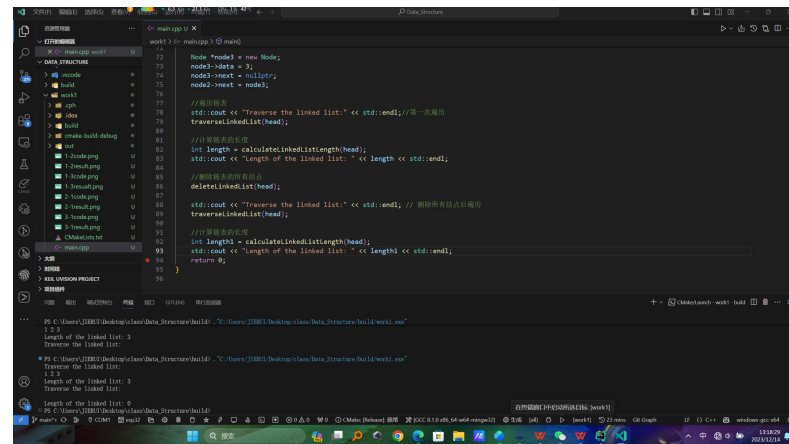


1. 单链表的建立:

RESULT:



```
1 // 1. 单链表的建立
2 #include <iostream>
3 using namespace std;
4 // 定义节点结构体
5 struct Node {
6     int data;
7     Node* next;
8 };
9 // 遍历链表
10 void TraverseList(Node* head) {
11     while (head != nullptr) {
12         cout << head->data << " ";
13         head = head->next;
14     }
15     cout << endl;
16 }
17 // 计算链表长度
18 int CalculateListLength(Node* head) {
19     int length = 0;
20     while (head != nullptr) {
21         length++;
22         head = head->next;
23     }
24     return length;
25 }
26 // 删除链表头节点
27 void DeleteFirstNode(Node*& head) {
28     if (head != nullptr) {
29         Node* newHead = head->next;
30         delete head;
31         head = newHead;
32     }
33 }
34 // 主函数
35 int main() {
36     Node* head = nullptr;
37     Node* node1 = new Node;
38     node1->data = 1;
39     node1->next = nullptr;
40     head = node1;
41     // 遍历链表
42     TraverseList(head);
43     // 计算链表长度
44     int length = CalculateListLength(head);
45     cout << "Length of the linked list: " << length << endl;
46     // 删除链表头节点
47     DeleteFirstNode(head);
48     TraverseList(head);
49     // 计算链表长度
50     int length1 = CalculateListLength(head);
51     cout << "Length of the linked list: " << length1 << endl;
52     return 0;
53 }
```

Output:

```
1 1
2 Length of the linked list: 1
3 Traverse the linked list:
4 1
5 Length of the linked list: 0
6 Traverse the linked list:
7 1 2 3
8 Length of the linked list: 3
9 Traverse the linked list:
10 2 3
11 Length of the linked list: 2
12 Traverse the linked list:
```

CODE:

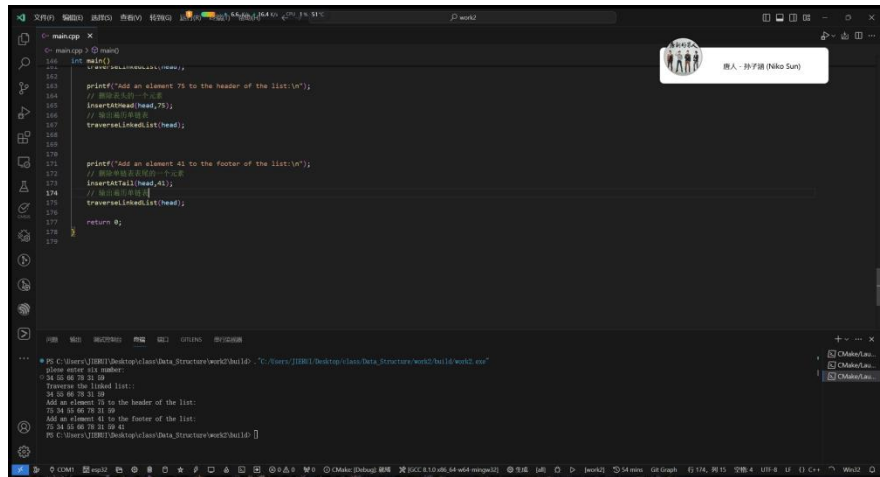
```

1 #include <iostream>
2 // 定义链表的结点结构体
3 struct Node
4 {
5     int data;
6     Node *next;
7 };
8 /**
9  * @brief 遍历链表
10  *
11  * @param head 链表头结点
12  */
13 void traverseLinkedList(Node *head)
14 {
15     Node *current = head;
16     while (current != nullptr) // 当前结点非空
17     {
18         std::cout << current->data << " ";
19         current = current->next;
20     }
21     std::cout << std::endl;
22 }
23 /**
24  * @brief 计算链表的长度
25  *
26  * @param head 链表头结点
27  * @return int 链表长度
28  */
29 int calculateLinkedListLength(Node *head)
30 {
31     int length = 0;
32     Node *current = head;
33     while (current != nullptr)
34     {
35         length++; // 长度自增
36         current = current->next; // 指针指向下一个结点
37     }
38     return length;
39 }
40 /**
41  * @brief 删除链表的所有结点
42  *
43  * @param head 链表头结点
44  */
45 void deleteLinkedList(Node *&head)
46 {
47     Node *current = head;
48     while (current != nullptr)
49     {
50         Node *temp = current;
51         current = current->next; // 指针指向下一个结点
52         delete temp;
53     }
54     head = nullptr;
55 }
56
57 int main()
58 {
59     Node *head = nullptr;
60
61     // 建立一个单链表
62     Node *node1 = new Node;
63     node1->data = 1;
64     node1->next = nullptr;
65     head = node1;
66
67     Node *node2 = new Node;
68     node2->data = 2;
69     node2->next = nullptr;
70     node1->next = node2;
71
72     Node *node3 = new Node;
73     node3->data = 3;
74     node3->next = nullptr;
75     node2->next = node3;
76
77     // 遍历链表
78     std::cout << "Traverse the linked list:" << std::endl; // 第一次遍历
79     traverseLinkedList(head);
80
81     // 计算链表的长度
82     int length = calculateLinkedListLength(head);
83     std::cout << "Length of the linked list: " << length << std::endl;
84
85     // 删除链表的所有结点
86     deleteLinkedList(head);
87
88     std::cout << "Traverse the linked list:" << std::endl; // 删除所有结点后遍历
89     traverseLinkedList(head);
90
91     // 计算链表的长度
92     int length1 = calculateLinkedListLength(head);
93     std::cout << "Length of the linked list: " << length1 << std::endl;
94     return 0;
95 }
96

```

2. 单链表的插入:

RESULT:



The screenshot shows a C++ IDE with a file named `main.cpp`. The code implements a linked list with `insertHead` and `insertTail` functions, and a `traverseLinkedList` function. The `main` function demonstrates adding elements 75 and 41 to the list and then traversing it.

```
144 int main()
145 {
146     LinkedList<int> list;
147     printf("Add an element 75 to the header of the list:\n");
148     // 向链表头部插入元素
149     insertHead(list, 75);
150     // 遍历链表
151     traverseLinkedList(list);
152
153     printf("Add an element 41 to the footer of the list:\n");
154     // 向链表尾部插入元素
155     insertTail(list, 41);
156     // 遍历链表
157     traverseLinkedList(list);
158
159     return 0;
160 }
```

The output window shows the execution results:

```
PS C:\Users\JERRY\Desktop\class_data_structures\work2\build> gcc -o work2.exe
Please enter six number:
24 88 75 31 28
Traverse the linked list:
24 88 75 31 28
Add an element 75 to the header of the list:
75 24 88 75 31 28
Add an element 41 to the footer of the list:
75 24 88 75 31 28 41
PS C:\Users\JERRY\Desktop\class_data_structures\work2\build>
```

CODE:

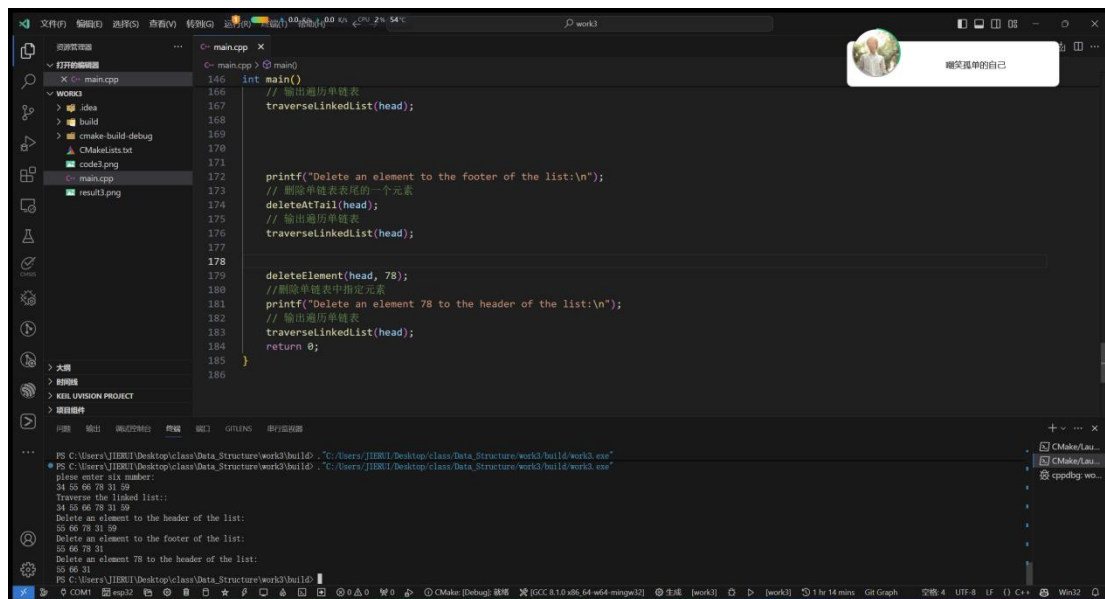
```

1 #include <iostream>
2 // 单链表节点结构体, 每一个节点包含数据和指向下一个节点的指针
3 struct Node
4 {
5     int data;
6     Node *next;
7 };
8 /**
9  * @brief 遍历单链表
10  *
11  * @param head 单链表的头指针
12  */
13 void traverseLinkedList(Node *head)
14 {
15     Node *current = head;
16     while (current != nullptr)
17     {
18         std::cout << current->data << " "; // 输出当前节点的数据, 并换行
19         current = current->next;           // 将current指向下一个节点
20     }
21     std::cout << std::endl; // 调用std::endl换行
22 }
23 /**
24  * @brief 向单链表的表头添加一个元素
25  *
26  * @param head 单链表的头指针
27  * @param value 待添加的元素
28  */
29 void insertAtHead(Node *head, int value)
30 {
31     Node *newNode = new Node();
32     newNode->data = value; // 将新节点的数据域赋值为value
33     newNode->next = head; // 将新节点的next指针指向原来的头节点
34     head = newNode;
35 }
36 /**
37  * @brief 向单链表的表尾添加一个元素
38  *
39  * @param head 单链表的头指针
40  * @param value 待添加的元素
41  */
42 void insertAtTail(Node *head, int value)
43 {
44     Node *newNode = new Node();
45     newNode->data = value; // 将新节点的数据域赋值为value
46     newNode->next = nullptr; // 将新节点的next指针置空
47
48     if (head == nullptr) // 如果为空链表, 直接将新节点称为头节点
49     {
50         head = newNode;
51     }
52     else
53     {
54         Node *current = head;
55         while (current->next != nullptr)
56         {
57             current = current->next;
58         }
59         current->next = newNode; // 将新节点添加到最后一个节点的后边
60     }
61 }
62 /**
63  * @brief 删除单链表的表头元素
64  *
65  * @param head 单链表的头指针
66  */
67 void deleteAtHead(Node *head)
68 {
69     if (head == nullptr) // 如果为空链表, 直接返回
70     {
71         return;
72     }
73
74     Node *temp = head; // 保存要删除节点的地址
75     head = head->next; // 头指针指向下一个节点
76     delete temp; // 释放原来的头节点
77 }
78 /**
79  * @brief 删除单链表的表尾元素
80  *
81  * @param head 单链表的头指针
82  */
83 void deleteAtTail(Node *head)
84 {
85     if (head == nullptr) // 如果为空链表, 直接返回
86     {
87         return;
88     }
89
90     if (head->next == nullptr) // 如果只有一个节点, 直接删除
91     {
92         delete head;
93         head = nullptr;
94         return;
95     }
96
97     Node *current = head;
98     while (current->next != nullptr) // 找到倒数第二个节点
99     {
100         current = current->next;
101     }
102     delete current->next; // 删除最后一个节点
103     current->next = nullptr; // 将倒数第二个节点的next指针置空
104 }
105 /**
106  * @brief 删除单链表中指定元素
107  *
108  * @param head 单链表的头指针
109  * @param value 待删除的元素
110  */
111 void deleteElement(Node *head, int value)
112 {
113     if (head == nullptr) // 如果为空链表, 直接返回
114     {
115         return;
116     }
117
118     // 如果要删除的元素是头节点
119     if (head->data == value)
120     {
121         deleteAtHead(head); // 直接删除头节点并返回
122         return;
123     }
124
125     Node *current = head; // 当前节点, 用于保存当前节点的地址
126     Node *previous = nullptr; // 前一个节点, 用于保存当前节点的前一个节点的地址
127
128     // 遍历单链表, 找到要删除的元素
129     while (current != nullptr && current->data != value) // 当链表不为空且当前节点的数据域不等于value, 继续向后遍历
130     {
131         previous = current;
132         current = current->next;
133     }
134
135     // 如果找到要删除的元素
136     if (current != nullptr)
137     {
138         previous->next = current->next; // 将当前节点的前一个节点的next指针指向当前节点的后一个节点
139         delete current; // 删除当前节点, 即删除指定元素
140     }
141 }
142
143 int main()
144 {
145     Node *head = nullptr; // 单链表的头指针, 初始化为nullptr
146
147     // 从键盘上读取6个整数, 并依次插入到单链表中
148     print("Please enter six number:\n");
149     for (int i = 0; i < 6; i++)
150     {
151         int num;
152         std::cin >> num;
153         insertAtTail(head, num); // 在单链表的表尾添加元素
154     }
155
156     // 遍历单链表
157     print("Traverse the linked list:\n");
158     traverseLinkedList(head);
159
160     print("Add an element 75 to the header of the list:\n");
161     // 向头结点添加一个元素
162     insertAtHead(head, 75);
163     // 再次遍历单链表
164     traverseLinkedList(head);
165
166     print("Add an element 41 to the footer of the list:\n");
167     // 向尾结点添加一个元素
168     insertAtTail(head, 41);
169     // 再次遍历单链表
170     traverseLinkedList(head);
171
172     return 0;
173 }

```

3. 单链表的删除:

RESULT:



```
145 int main()
146 {
147     // 输出遍历单链表
148     traverseLinkedList(head);
149
150     printf("Delete an element to the footer of the list:\n");
151     // 删除单链表表尾的一个元素
152     deleteAtTail(head);
153     // 输出遍历单链表
154     traverseLinkedList(head);
155
156     deleteElement(head, 78);
157     // 删除单链表中指定元素
158     printf("Delete an element 78 to the header of the list:\n");
159     // 输出遍历单链表
160     traverseLinkedList(head);
161     return 0;
162 }
```

PS C:\Users\JHERU\Desktop\class\Data_Structure\work3\build> . "C:\Users\JHERU\Desktop\class\Data_Structure\work3\build\work3.exe"

PS C:\Users\JHERU\Desktop\class\Data_Structure\work3\build> . "C:\Users\JHERU\Desktop\class\Data_Structure\work3\build\work3.exe"

please enter a/n number:

34 55 66 78 31 59

Traverse the linked list::

34 55 66 78 31 59

Delete an element to the footer of the list:

55 66 78 31 59

Delete an element to the footer of the list:

55 66 78 31

Delete an element 78 to the header of the list:

55 66 31

PS C:\Users\JHERU\Desktop\class\Data_Structure\work3\build>

CODE:

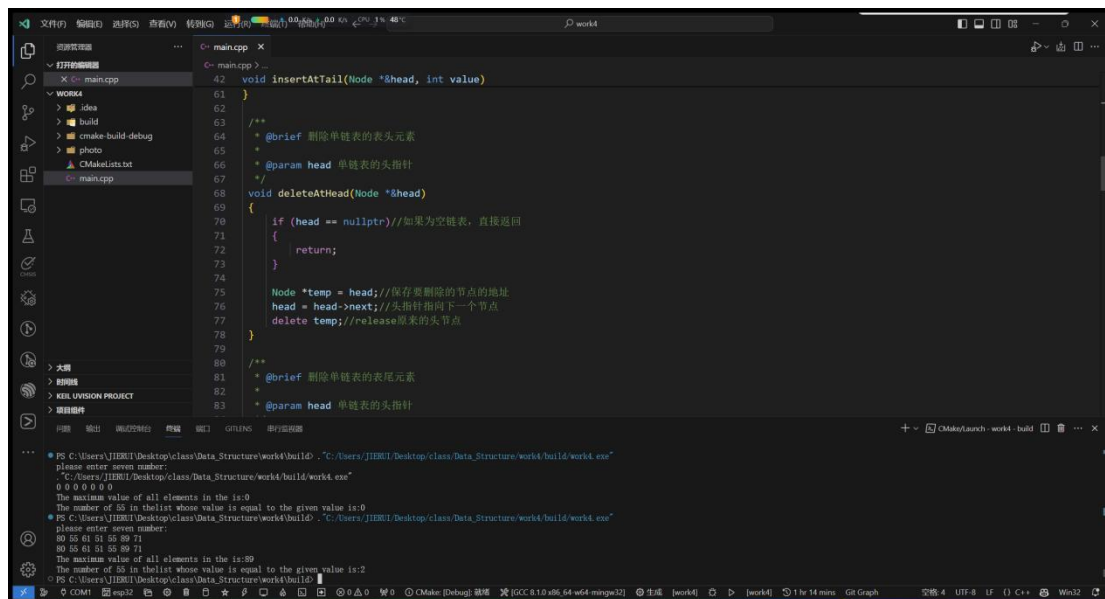
```

1 #include <iostream>
2 // 单链表的节点结构体,即一个节点包含数据和指向下一个节点的指针
3 struct Node
4 {
5     int data;
6     Node *next;
7 };
8 /**
9  * @brief 遍历单链表
10 */
11 // @param head 存储表头指针
12 //
13 void traverseLinkedList(Node *head)
14 {
15     Node *current = head;
16     while (current != nullptr)
17     {
18         std::cout << current->data << " "; // 输出当前节点的数据,并换
19         current = current->next; // 将current指向下一个节点
20     }
21     std::cout << std::endl; // 调用std::endl换行
22 }
23 /**
24  * @brief 向单链表的表头添加一个元素
25 */
26 // @param head 存储表头指针
27 // @param value 要添加的元素
28 //
29 void insertAtHead(Node *head, int value)
30 {
31     Node *newNode = new Node();
32     newNode->data = value; // 给新节点的数据域赋值value
33     newNode->next = head; // 将新节点的next指针指向原来的头节点
34     head = newNode;
35 }
36 /**
37  * @brief 向单链表的表尾添加一个元素
38 */
39 // @param head 存储表头指针
40 // @param value 要添加的元素
41 //
42 void insertAtTail(Node *head, int value)
43 {
44     Node *newNode = new Node();
45     newNode->data = value; // 给新节点的数据域赋值value
46     newNode->next = nullptr; // 将新节点的next指针置空
47
48     if (head == nullptr) // 如果为空链表,直接将新节点作为头节点
49     {
50         head = newNode;
51     }
52     else
53     {
54         Node *current = head;
55         while (current->next != nullptr)
56         {
57             current = current->next;
58         }
59         current->next = newNode; // 将新节点添加到最后一个节点的后面
60     }
61 }
62 /**
63  * @brief 删除单链表的表头元素
64 */
65 // @param head 存储表头指针
66 //
67 void deleteAtHead(Node *head)
68 {
69     if (head == nullptr) // 如果为空链表,直接返回
70     {
71         return;
72     }
73
74     Node *temp = head; // 保存要删除的节点的地址
75     head = head->next; // 头指针指向下一个节点
76     delete temp; // 释放原来的头节点
77 }
78 /**
79  * @brief 删除单链表的表尾元素
80 */
81 // @param head 存储表头指针
82 //
83 void deleteAtTail(Node *head)
84 {
85     if (head == nullptr) // 如果为空链表,直接返回
86     {
87         return;
88     }
89
90     if (head->next == nullptr) // 如果只有一个节点,直接删除
91     {
92         delete head;
93         head = nullptr;
94         return;
95     }
96
97     Node *current = head;
98     while (current->next->next != nullptr) // 找到倒数第二个节点
99     {
100         current = current->next;
101     }
102     delete current->next; // 删除最后一个节点
103     current->next = nullptr; // 倒数第二个节点的next指针置空
104 }
105 /**
106  * @brief 删除单链表中指定元素
107 */
108 // @param head 存储表头指针
109 // @param value 要删除的元素
110 //
111 void deleteElement(Node *head, int value)
112 {
113     if (head == nullptr) // 如果为空链表,直接返回
114     {
115         return;
116     }
117
118     // 如果头节点的元素是目标元素
119     if (head->data == value)
120     {
121         deleteAtHead(head);
122         return;
123     }
124
125     Node *current = head;
126     Node *previous = nullptr;
127
128     // 遍历链表寻找要删除的元素
129     while (current != nullptr && current->data != value)
130     {
131         previous = current;
132         current = current->next;
133     }
134
135     // 如果找到要删除的元素
136     if (current != nullptr)
137     {
138         previous->next = current->next;
139         delete current;
140     }
141 }
142
143 int main()
144 {
145     Node *head = nullptr; // 单链表的头指针,初始化为nullptr
146
147     // 从键盘上输入六个整数,并依次添加到单链表中
148     printf("Please enter six number:\n");
149     for (int i = 0; i < 6; i++)
150     {
151         int num;
152         std::cin >> num;
153         insertAtHead(head, num); // 在单链表的表头添加元素
154     }
155
156     // 输出单链表的表头
157     printf("Traverse the linked list:\n");
158     traverseLinkedList(head);
159
160     printf("Delete an element to the header of the list:\n");
161     // 删除头节点(表头)
162     deleteAtHead(head);
163     // 输出单链表的表头
164     traverseLinkedList(head);
165
166     printf("Delete an element to the footer of the list:\n");
167     // 删除尾节点(表尾)
168     deleteAtTail(head);
169     // 输出单链表的表头
170     traverseLinkedList(head);
171
172     printf("Delete an element 78 to the header of the list:\n");
173     // 删除单链表中指定元素
174     deleteElement(head, 78);
175     // 输出单链表的表头
176     printf("Traverse the linked list:\n");
177     traverseLinkedList(head);
178     return 0;
179 }

```

4. 单链表的查找:

RESULT:



The screenshot shows a C++ IDE with a project named 'work4'. The main.cpp file contains the following code:

```
42 void insertAtTail(Node *head, int value)
43 {
44 }
45
46 /**
47  * @brief 删除单链表的表头元素
48  *
49  * @param head 单链表的头指针
50  */
51 void deleteAtHead(Node *head)
52 {
53     if (head == nullptr) // 如果为空链表，直接返回
54     {
55         return;
56     }
57     Node *temp = head; // 保存要删除的节点的地址
58     head = head->next; // 头指针指向下一个节点
59     delete temp; // release原来的头节点
60 }
61
62 /**
63  * @brief 删除单链表的表尾元素
64  *
65  * @param head 单链表的头指针
66  */
```

The output window shows the following execution results:

```
PS C:\Users\JHERU\Desktop\class\Data_Structure\work4\build> . "C:\Users\JHERU\Desktop\class\Data_Structure\work4\build\work4.exe"
please enter seven number:
0 0 0 0 0 0
The maximum value of all elements in the list is:0
The number of 55 in the list whose value is equal to the given value is:0
please enter seven number:
80 55 61 51 55 89 71
The maximum value of all elements in the list is:89
The number of 55 in the list whose value is equal to the given value is:2
PS C:\Users\JHERU\Desktop\class\Data_Structure\work4\build>
```

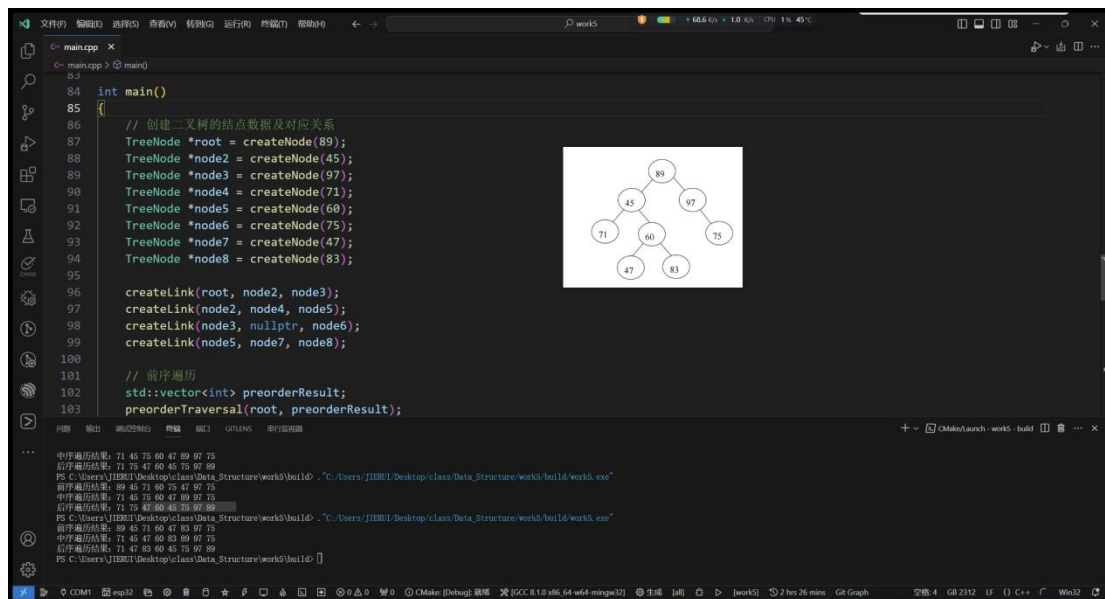
CODE:

```

1 // 双向链表
2 // 双向链表的特点是，每个节点包含数据域和两个指针域，一个指向下一个节点，一个指向上一个节点
3 struct Node
4 {
5     int data;
6     Node *next;
7     Node *prev;
8 };
9
10 // 初始化双向链表
11 // 双向链表头指针
12 Node *head = NULL;
13 // 双向链表尾指针
14 Node *tail = NULL;
15 // 双向链表当前节点指针
16 Node *current = NULL;
17 // 双向链表前一个节点指针
18 Node *prev = NULL;
19 // 双向链表后一个节点指针
20 Node *next = NULL;
21 // 双向链表节点数
22 int count = 0;
23
24 // 双向链表插入节点
25 // 在双向链表头部插入节点
26 void insertHead(int value)
27 {
28     Node *newNode = new Node();
29     newNode->data = value;
30     newNode->next = head;
31     newNode->prev = NULL;
32     head = newNode;
33     if (tail == NULL)
34     {
35         tail = newNode;
36     }
37     current = head;
38     while (current->next != NULL)
39     {
40         current = current->next;
41     }
42     current->next = newNode;
43 }
44
45 // 在双向链表尾部插入节点
46 void insertTail(int value)
47 {
48     Node *newNode = new Node();
49     newNode->data = value;
50     newNode->prev = tail;
51     newNode->next = NULL;
52     tail = newNode;
53     if (head == NULL)
54     {
55         head = newNode;
56     }
57     current = tail;
58     while (current->prev != NULL)
59     {
60         current = current->prev;
61     }
62     current->prev = newNode;
63 }
64
65 // 在双向链表中间插入节点
66 void insertNode(int value, int index)
67 {
68     Node *newNode = new Node();
69     newNode->data = value;
70     newNode->prev = NULL;
71     newNode->next = NULL;
72     if (index == 1)
73     {
74         insertHead(value);
75     }
76     else if (index == count + 1)
77     {
78         insertTail(value);
79     }
80     else
81     {
82         current = head;
83         while (current->next != NULL)
84         {
85             current = current->next;
86             index--;
87         }
88         if (index == 1)
89         {
90             insertHead(value);
91         }
92         else
93         {
94             current->next->prev = newNode;
95             newNode->next = current->next;
96             newNode->prev = current;
97             current->next = newNode;
98         }
99     }
100 }
101
102 // 双向链表删除节点
103 void deleteNode(int value)
104 {
105     Node *temp = head;
106     while (temp->next != NULL)
107     {
108         if (temp->data == value)
109         {
110             delete temp;
111             if (temp == head)
112             {
113                 head = temp->next;
114             }
115             if (temp == tail)
116             {
117                 tail = temp->prev;
118             }
119             if (temp == current)
120             {
121                 current = temp->next;
122             }
123             if (temp == prev)
124             {
125                 prev = temp->prev;
126             }
127             count--;
128         }
129         temp = temp->next;
130     }
131 }
132
133 // 双向链表查找节点
134 Node *findNode(int value)
135 {
136     Node *temp = head;
137     while (temp->next != NULL)
138     {
139         if (temp->data == value)
140         {
141             return temp;
142         }
143         temp = temp->next;
144     }
145     return NULL;
146 }
147
148 // 双向链表最大值
149 int findMaxValue()
150 {
151     Node *temp = head;
152     int max = temp->data;
153     while (temp->next != NULL)
154     {
155         if (temp->data > max)
156         {
157             max = temp->data;
158         }
159         temp = temp->next;
160     }
161     return max;
162 }
163
164 // 双向链表最小值
165 int findMinValue()
166 {
167     Node *temp = head;
168     int min = temp->data;
169     while (temp->next != NULL)
170     {
171         if (temp->data < min)
172         {
173             min = temp->data;
174         }
175         temp = temp->next;
176     }
177     return min;
178 }
179
180 // 双向链表反转
181 void reverse()
182 {
183     Node *temp = head;
184     Node *newHead = NULL;
185     while (temp != NULL)
186     {
187         Node *newNode = new Node();
188         newNode->data = temp->data;
189         newNode->next = newHead;
190         newHead = newNode;
191         temp = temp->next;
192     }
193     head = newHead;
194     tail = temp;
195 }
196
197 // 双向链表排序
198 void sort()
199 {
200     Node *temp = head;
201     Node *newHead = NULL;
202     while (temp != NULL)
203     {
204         Node *newNode = new Node();
205         newNode->data = temp->data;
206         newNode->next = newHead;
207         newHead = newNode;
208         temp = temp->next;
209     }
210     head = newHead;
211     tail = temp;
212 }
213
214 // 双向链表合并
215 void merge()
216 {
217     Node *temp1 = head1;
218     Node *temp2 = head2;
219     Node *newHead = NULL;
220     while (temp1 != NULL || temp2 != NULL)
221     {
222         if (temp1->data < temp2->data)
223         {
224             Node *newNode = new Node();
225             newNode->data = temp1->data;
226             newNode->next = newHead;
227             newHead = newNode;
228             temp1 = temp1->next;
229         }
230         else
231         {
232             Node *newNode = new Node();
233             newNode->data = temp2->data;
234             newNode->next = newHead;
235             newHead = newNode;
236             temp2 = temp2->next;
237         }
238     }
239     head = newHead;
240     tail = temp1 == NULL ? temp2 : temp1;
241 }
242
243 // 双向链表求和
244 int sum()
245 {
246     Node *temp = head;
247     int sum = 0;
248     while (temp != NULL)
249     {
250         sum += temp->data;
251         temp = temp->next;
252     }
253     return sum;
254 }
255
256 // 双向链表求平均
257 double avg()
258 {
259     Node *temp = head;
260     int sum = 0;
261     int count = 0;
262     while (temp != NULL)
263     {
264         sum += temp->data;
265         count++;
266         temp = temp->next;
267     }
268     return sum / count;
269 }
270
271 // 双向链表求方差
272 double var()
273 {
274     Node *temp = head;
275     int sum = 0;
276     int count = 0;
277     while (temp != NULL)
278     {
279         sum += temp->data;
280         count++;
281         temp = temp->next;
282     }
283     double avg = avg();
284     double sum2 = 0;
285     temp = head;
286     while (temp != NULL)
287     {
288         sum2 += (temp->data - avg) * (temp->data - avg);
289         temp = temp->next;
290     }
291     return sum2 / count;
292 }
293
294 // 双向链表求标准差
295 double std()
296 {
297     return sqrt(var());
298 }
299
300 // 双向链表求协方差
301 double cov()
302 {
303     Node *temp1 = head1;
304     Node *temp2 = head2;
305     int sum = 0;
306     int count1 = 0;
307     int count2 = 0;
308     while (temp1 != NULL || temp2 != NULL)
309     {
310         if (temp1 != NULL)
311         {
312             sum += temp1->data;
313             count1++;
314             temp1 = temp1->next;
315         }
316         if (temp2 != NULL)
317         {
318             sum += temp2->data;
319             count2++;
320             temp2 = temp2->next;
321         }
322     }
323     return sum / (count1 * count2);
324 }
325
326 // 双向链表求相关系数
327 double cor()
328 {
329     double cov = cov();
330     double std1 = std();
331     double std2 = std();
332     return cov / (std1 * std2);
333 }
334
335 // 双向链表求回归方程
336 void regression()
337 {
338     Node *temp = head;
339     int sumX = 0;
340     int sumY = 0;
341     int countX = 0;
342     int countY = 0;
343     while (temp != NULL)
344     {
345         sumX += temp->data;
346         sumY += temp->data;
347         countX++;
348         countY++;
349         temp = temp->next;
350     }
351     double avgX = sumX / countX;
352     double avgY = sumY / countY;
353     double cov = cov();
354     double stdX = std();
355     double stdY = std();
356     double cor = cor();
357     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
358     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
359     printf("The regression equation is: y = %f * x + %f\n", a, b);
360 }
361
362 // 双向链表求主成分分析
363 void pca()
364 {
365     Node *temp = head;
366     int sumX = 0;
367     int sumY = 0;
368     int countX = 0;
369     int countY = 0;
370     while (temp != NULL)
371     {
372         sumX += temp->data;
373         sumY += temp->data;
374         countX++;
375         countY++;
376         temp = temp->next;
377     }
378     double avgX = sumX / countX;
379     double avgY = sumY / countY;
380     double cov = cov();
381     double stdX = std();
382     double stdY = std();
383     double cor = cor();
384     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
385     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
386     printf("The principal component analysis result is: y = %f * x + %f\n", a, b);
387 }
388
389 // 双向链表求聚类分析
390 void cluster()
391 {
392     Node *temp = head;
393     int sumX = 0;
394     int sumY = 0;
395     int countX = 0;
396     int countY = 0;
397     while (temp != NULL)
398     {
399         sumX += temp->data;
400         sumY += temp->data;
401         countX++;
402         countY++;
403         temp = temp->next;
404     }
405     double avgX = sumX / countX;
406     double avgY = sumY / countY;
407     double cov = cov();
408     double stdX = std();
409     double stdY = std();
410     double cor = cor();
411     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
412     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
413     printf("The cluster analysis result is: y = %f * x + %f\n", a, b);
414 }
415
416 // 双向链表求决策树
417 void decisionTree()
418 {
419     Node *temp = head;
420     int sumX = 0;
421     int sumY = 0;
422     int countX = 0;
423     int countY = 0;
424     while (temp != NULL)
425     {
426         sumX += temp->data;
427         sumY += temp->data;
428         countX++;
429         countY++;
430         temp = temp->next;
431     }
432     double avgX = sumX / countX;
433     double avgY = sumY / countY;
434     double cov = cov();
435     double stdX = std();
436     double stdY = std();
437     double cor = cor();
438     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
439     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
440     printf("The decision tree result is: y = %f * x + %f\n", a, b);
441 }
442
443 // 双向链表求神经网络
444 void neuralNetwork()
445 {
446     Node *temp = head;
447     int sumX = 0;
448     int sumY = 0;
449     int countX = 0;
450     int countY = 0;
451     while (temp != NULL)
452     {
453         sumX += temp->data;
454         sumY += temp->data;
455         countX++;
456         countY++;
457         temp = temp->next;
458     }
459     double avgX = sumX / countX;
460     double avgY = sumY / countY;
461     double cov = cov();
462     double stdX = std();
463     double stdY = std();
464     double cor = cor();
465     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
466     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
467     printf("The neural network result is: y = %f * x + %f\n", a, b);
468 }
469
470 // 双向链表求支持向量机
471 void svm()
472 {
473     Node *temp = head;
474     int sumX = 0;
475     int sumY = 0;
476     int countX = 0;
477     int countY = 0;
478     while (temp != NULL)
479     {
480         sumX += temp->data;
481         sumY += temp->data;
482         countX++;
483         countY++;
484         temp = temp->next;
485     }
486     double avgX = sumX / countX;
487     double avgY = sumY / countY;
488     double cov = cov();
489     double stdX = std();
490     double stdY = std();
491     double cor = cor();
492     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
493     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
494     printf("The svm result is: y = %f * x + %f\n", a, b);
495 }
496
497 // 双向链表求随机森林
498 void randomForest()
499 {
500     Node *temp = head;
501     int sumX = 0;
502     int sumY = 0;
503     int countX = 0;
504     int countY = 0;
505     while (temp != NULL)
506     {
507         sumX += temp->data;
508         sumY += temp->data;
509         countX++;
510         countY++;
511         temp = temp->next;
512     }
513     double avgX = sumX / countX;
514     double avgY = sumY / countY;
515     double cov = cov();
516     double stdX = std();
517     double stdY = std();
518     double cor = cor();
519     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
520     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
521     printf("The random forest result is: y = %f * x + %f\n", a, b);
522 }
523
524 // 双向链表求梯度下降法
525 void gradientDescent()
526 {
527     Node *temp = head;
528     int sumX = 0;
529     int sumY = 0;
530     int countX = 0;
531     int countY = 0;
532     while (temp != NULL)
533     {
534         sumX += temp->data;
535         sumY += temp->data;
536         countX++;
537         countY++;
538         temp = temp->next;
539     }
540     double avgX = sumX / countX;
541     double avgY = sumY / countY;
542     double cov = cov();
543     double stdX = std();
544     double stdY = std();
545     double cor = cor();
546     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
547     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
548     printf("The gradient descent result is: y = %f * x + %f\n", a, b);
549 }
550
551 // 双向链表求遗传算法
552 void geneticAlgorithm()
553 {
554     Node *temp = head;
555     int sumX = 0;
556     int sumY = 0;
557     int countX = 0;
558     int countY = 0;
559     while (temp != NULL)
560     {
561         sumX += temp->data;
562         sumY += temp->data;
563         countX++;
564         countY++;
565         temp = temp->next;
566     }
567     double avgX = sumX / countX;
568     double avgY = sumY / countY;
569     double cov = cov();
570     double stdX = std();
571     double stdY = std();
572     double cor = cor();
573     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
574     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
575     printf("The genetic algorithm result is: y = %f * x + %f\n", a, b);
576 }
577
578 // 双向链表求模拟退火算法
579 void simulatedAnnealing()
580 {
581     Node *temp = head;
582     int sumX = 0;
583     int sumY = 0;
584     int countX = 0;
585     int countY = 0;
586     while (temp != NULL)
587     {
588         sumX += temp->data;
589         sumY += temp->data;
590         countX++;
591         countY++;
592         temp = temp->next;
593     }
594     double avgX = sumX / countX;
595     double avgY = sumY / countY;
596     double cov = cov();
597     double stdX = std();
598     double stdY = std();
599     double cor = cor();
600     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
601     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
602     printf("The simulated annealing result is: y = %f * x + %f\n", a, b);
603 }
604
605 // 双向链表求粒子群优化算法
606 void particleSwarmOptimization()
607 {
608     Node *temp = head;
609     int sumX = 0;
610     int sumY = 0;
611     int countX = 0;
612     int countY = 0;
613     while (temp != NULL)
614     {
615         sumX += temp->data;
616         sumY += temp->data;
617         countX++;
618         countY++;
619         temp = temp->next;
620     }
621     double avgX = sumX / countX;
622     double avgY = sumY / countY;
623     double cov = cov();
624     double stdX = std();
625     double stdY = std();
626     double cor = cor();
627     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
628     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
629     printf("The particle swarm optimization result is: y = %f * x + %f\n", a, b);
630 }
631
632 // 双向链表求蚁群优化算法
633 void antColonyOptimization()
634 {
635     Node *temp = head;
636     int sumX = 0;
637     int sumY = 0;
638     int countX = 0;
639     int countY = 0;
640     while (temp != NULL)
641     {
642         sumX += temp->data;
643         sumY += temp->data;
644         countX++;
645         countY++;
646         temp = temp->next;
647     }
648     double avgX = sumX / countX;
649     double avgY = sumY / countY;
650     double cov = cov();
651     double stdX = std();
652     double stdY = std();
653     double cor = cor();
654     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
655     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
656     printf("The ant colony optimization result is: y = %f * x + %f\n", a, b);
657 }
658
659 // 双向链表求禁忌搜索算法
660 void tabuSearch()
661 {
662     Node *temp = head;
663     int sumX = 0;
664     int sumY = 0;
665     int countX = 0;
666     int countY = 0;
667     while (temp != NULL)
668     {
669         sumX += temp->data;
670         sumY += temp->data;
671         countX++;
672         countY++;
673         temp = temp->next;
674     }
675     double avgX = sumX / countX;
676     double avgY = sumY / countY;
677     double cov = cov();
678     double stdX = std();
679     double stdY = std();
680     double cor = cor();
681     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
682     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
683     printf("The tabu search result is: y = %f * x + %f\n", a, b);
684 }
685
686 // 双向链表求遗传编程算法
687 void geneticProgramming()
688 {
689     Node *temp = head;
690     int sumX = 0;
691     int sumY = 0;
692     int countX = 0;
693     int countY = 0;
694     while (temp != NULL)
695     {
696         sumX += temp->data;
697         sumY += temp->data;
698         countX++;
699         countY++;
700         temp = temp->next;
701     }
702     double avgX = sumX / countX;
703     double avgY = sumY / countY;
704     double cov = cov();
705     double stdX = std();
706     double stdY = std();
707     double cor = cor();
708     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
709     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
710     printf("The genetic programming result is: y = %f * x + %f\n", a, b);
711 }
712
713 // 双向链表求进化策略算法
714 void evolutionaryStrategy()
715 {
716     Node *temp = head;
717     int sumX = 0;
718     int sumY = 0;
719     int countX = 0;
720     int countY = 0;
721     while (temp != NULL)
722     {
723         sumX += temp->data;
724         sumY += temp->data;
725         countX++;
726         countY++;
727         temp = temp->next;
728     }
729     double avgX = sumX / countX;
730     double avgY = sumY / countY;
731     double cov = cov();
732     double stdX = std();
733     double stdY = std();
734     double cor = cor();
735     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
736     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
737     printf("The evolutionary strategy result is: y = %f * x + %f\n", a, b);
738 }
739
740 // 双向链表求差分进化算法
741 void differentialEvolution()
742 {
743     Node *temp = head;
744     int sumX = 0;
745     int sumY = 0;
746     int countX = 0;
747     int countY = 0;
748     while (temp != NULL)
749     {
750         sumX += temp->data;
751         sumY += temp->data;
752         countX++;
753         countY++;
754         temp = temp->next;
755     }
756     double avgX = sumX / countX;
757     double avgY = sumY / countY;
758     double cov = cov();
759     double stdX = std();
760     double stdY = std();
761     double cor = cor();
762     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
763     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
764     printf("The differential evolution result is: y = %f * x + %f\n", a, b);
765 }
766
767 // 双向链表求鲸鱼优化算法
768 void whaleOptimization()
769 {
770     Node *temp = head;
771     int sumX = 0;
772     int sumY = 0;
773     int countX = 0;
774     int countY = 0;
775     while (temp != NULL)
776     {
777         sumX += temp->data;
778         sumY += temp->data;
779         countX++;
780         countY++;
781         temp = temp->next;
782     }
783     double avgX = sumX / countX;
784     double avgY = sumY / countY;
785     double cov = cov();
786     double stdX = std();
787     double stdY = std();
788     double cor = cor();
789     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
790     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
791     printf("The whale optimization result is: y = %f * x + %f\n", a, b);
792 }
793
794 // 双向链表求灰狼优化算法
795 void greyWolfOptimization()
796 {
797     Node *temp = head;
798     int sumX = 0;
799     int sumY = 0;
800     int countX = 0;
801     int countY = 0;
802     while (temp != NULL)
803     {
804         sumX += temp->data;
805         sumY += temp->data;
806         countX++;
807         countY++;
808         temp = temp->next;
809     }
810     double avgX = sumX / countX;
811     double avgY = sumY / countY;
812     double cov = cov();
813     double stdX = std();
814     double stdY = std();
815     double cor = cor();
816     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
817     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
818     printf("The grey wolf optimization result is: y = %f * x + %f\n", a, b);
819 }
820
821 // 双向链表求狮子群优化算法
822 void lionGroupOptimization()
823 {
824     Node *temp = head;
825     int sumX = 0;
826     int sumY = 0;
827     int countX = 0;
828     int countY = 0;
829     while (temp != NULL)
830     {
831         sumX += temp->data;
832         sumY += temp->data;
833         countX++;
834         countY++;
835         temp = temp->next;
836     }
837     double avgX = sumX / countX;
838     double avgY = sumY / countY;
839     double cov = cov();
840     double stdX = std();
841     double stdY = std();
842     double cor = cor();
843     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
844     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
845     printf("The lion group optimization result is: y = %f * x + %f\n", a, b);
846 }
847
848 // 双向链表求布谷鸟优化算法
849 void cuckooOptimization()
850 {
851     Node *temp = head;
852     int sumX = 0;
853     int sumY = 0;
854     int countX = 0;
855     int countY = 0;
856     while (temp != NULL)
857     {
858         sumX += temp->data;
859         sumY += temp->data;
860         countX++;
861         countY++;
862         temp = temp->next;
863     }
864     double avgX = sumX / countX;
865     double avgY = sumY / countY;
866     double cov = cov();
867     double stdX = std();
868     double stdY = std();
869     double cor = cor();
870     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
871     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
872     printf("The cuckoo optimization result is: y = %f * x + %f\n", a, b);
873 }
874
875 // 双向链表求萤火虫优化算法
876 void fireflyOptimization()
877 {
878     Node *temp = head;
879     int sumX = 0;
880     int sumY = 0;
881     int countX = 0;
882     int countY = 0;
883     while (temp != NULL)
884     {
885         sumX += temp->data;
886         sumY += temp->data;
887         countX++;
888         countY++;
889         temp = temp->next;
890     }
891     double avgX = sumX / countX;
892     double avgY = sumY / countY;
893     double cov = cov();
894     double stdX = std();
895     double stdY = std();
896     double cor = cor();
897     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
898     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
899     printf("The firefly optimization result is: y = %f * x + %f\n", a, b);
900 }
901
902 // 双向链表求粒子群优化算法
903 void particleSwarmOptimization2()
904 {
905     Node *temp = head;
906     int sumX = 0;
907     int sumY = 0;
908     int countX = 0;
909     int countY = 0;
910     while (temp != NULL)
911     {
912         sumX += temp->data;
913         sumY += temp->data;
914         countX++;
915         countY++;
916         temp = temp->next;
917     }
918     double avgX = sumX / countX;
919     double avgY = sumY / countY;
920     double cov = cov();
921     double stdX = std();
922     double stdY = std();
923     double cor = cor();
924     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
925     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
926     printf("The particle swarm optimization result is: y = %f * x + %f\n", a, b);
927 }
928
929 // 双向链表求蚁群优化算法
930 void antColonyOptimization2()
931 {
932     Node *temp = head;
933     int sumX = 0;
934     int sumY = 0;
935     int countX = 0;
936     int countY = 0;
937     while (temp != NULL)
938     {
939         sumX += temp->data;
940         sumY += temp->data;
941         countX++;
942         countY++;
943         temp = temp->next;
944     }
945     double avgX = sumX / countX;
946     double avgY = sumY / countY;
947     double cov = cov();
948     double stdX = std();
949     double stdY = std();
950     double cor = cor();
951     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
952     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
953     printf("The ant colony optimization result is: y = %f * x + %f\n", a, b);
954 }
955
956 // 双向链表求禁忌搜索算法
957 void tabuSearch2()
958 {
959     Node *temp = head;
960     int sumX = 0;
961     int sumY = 0;
962     int countX = 0;
963     int countY = 0;
964     while (temp != NULL)
965     {
966         sumX += temp->data;
967         sumY += temp->data;
968         countX++;
969         countY++;
970         temp = temp->next;
971     }
972     double avgX = sumX / countX;
973     double avgY = sumY / countY;
974     double cov = cov();
975     double stdX = std();
976     double stdY = std();
977     double cor = cor();
978     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
979     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
980     printf("The tabu search result is: y = %f * x + %f\n", a, b);
981 }
982
983 // 双向链表求遗传编程算法
984 void geneticProgramming2()
985 {
986     Node *temp = head;
987     int sumX = 0;
988     int sumY = 0;
989     int countX = 0;
990     int countY = 0;
991     while (temp != NULL)
992     {
993         sumX += temp->data;
994         sumY += temp->data;
995         countX++;
996         countY++;
997         temp = temp->next;
998     }
999     double avgX = sumX / countX;
1000    double avgY = sumY / countY;
1001    double cov = cov();
1002    double stdX = std();
1003    double stdY = std();
1004    double cor = cor();
1005    double a = (cov - cor * stdX * stdY) / (stdX * stdX);
1006    double b = (cov - cor * stdX * stdY) / (stdY * stdY);
1007    printf("The genetic programming result is: y = %f * x + %f\n", a, b);
1008 }
1009
1010 // 双向链表求进化策略算法
1011 void evolutionaryStrategy2()
1012 {
1013     Node *temp = head;
1014     int sumX = 0;
1015     int sumY = 0;
1016     int countX = 0;
1017     int countY = 0;
1018     while (temp != NULL)
1019     {
1020         sumX += temp->data;
1021         sumY += temp->data;
1022         countX++;
1023         countY++;
1024         temp = temp->next;
1025     }
1026     double avgX = sumX / countX;
1027     double avgY = sumY / countY;
1028     double cov = cov();
1029     double stdX = std();
1030     double stdY = std();
1031     double cor = cor();
1032     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
1033     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
1034     printf("The evolutionary strategy result is: y = %f * x + %f\n", a, b);
1035 }
1036
1037 // 双向链表求差分进化算法
1038 void differentialEvolution2()
1039 {
1040     Node *temp = head;
1041     int sumX = 0;
1042     int sumY = 0;
1043     int countX = 0;
1044     int countY = 0;
1045     while (temp != NULL)
1046     {
1047         sumX += temp->data;
1048         sumY += temp->data;
1049         countX++;
1050         countY++;
1051         temp = temp->next;
1052     }
1053     double avgX = sumX / countX;
1054     double avgY = sumY / countY;
1055     double cov = cov();
1056     double stdX = std();
1057     double stdY = std();
1058     double cor = cor();
1059     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
1060     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
1061     printf("The differential evolution result is: y = %f * x + %f\n", a, b);
1062 }
1063
1064 // 双向链表求鲸鱼优化算法
1065 void whaleOptimization2()
1066 {
1067     Node *temp = head;
1068     int sumX = 0;
1069     int sumY = 0;
1070     int countX = 0;
1071     int countY = 0;
1072     while (temp != NULL)
1073     {
1074         sumX += temp->data;
1075         sumY += temp->data;
1076         countX++;
1077         countY++;
1078         temp = temp->next;
1079     }
1080     double avgX = sumX / countX;
1081     double avgY = sumY / countY;
1082     double cov = cov();
1083     double stdX = std();
1084     double stdY = std();
1085     double cor = cor();
1086     double a = (cov - cor * stdX * stdY) / (stdX * stdX);
1087     double b = (cov - cor * stdX * stdY) / (stdY * stdY);
1088     printf("The whale optimization result is: y = %f * x + %f\n", a, b);
1089 }
1090
1091 // 双向链表求灰狼优化算法
1092 void greyWolfOptimization2()
1093
```


5. 树的操作与处理:

RESULT:



The screenshot shows a C++ IDE with a file named `main.cpp`. The code defines a binary tree structure and performs a preorder traversal. The tree structure is as follows:

```
graph TD
    89((89)) --- 45((45))
    89 --- 97((97))
    45 --- 71((71))
    45 --- 60((60))
    60 --- 47((47))
    60 --- 83((83))
    75((75)) --- 97
```

The code in `main.cpp` is as follows:

```
1 // 创建二叉树的结点数据及对应关系
2
3 #include <iostream>
4 #include <vector>
5 #include <memory>
6
7 using namespace std;
8
9 struct TreeNode {
10     int val;
11     TreeNode *left;
12     TreeNode *right;
13 };
14
15 // 创建结点
16 TreeNode* createNode(int val) {
17     return new TreeNode(val);
18 }
19
20 // 创建二叉树
21 void createTree(TreeNode*& root, int val, int left, int right) {
22     if (val == -1) return;
23     root = createNode(val);
24     createTree(root->left, left, -1);
25     createTree(root->right, right, -1);
26 }
27
28 // 前序遍历
29 void preorderTraversal(TreeNode* root, vector<int>& result) {
30     if (root == nullptr) return;
31     result.push_back(root->val);
32     preorderTraversal(root->left, result);
33     preorderTraversal(root->right, result);
34 }
35
36 int main() {
37     // 创建二叉树的结点数据及对应关系
38     TreeNode* root = createNode(89);
39     TreeNode* node2 = createNode(45);
40     TreeNode* node3 = createNode(97);
41     TreeNode* node4 = createNode(71);
42     TreeNode* node5 = createNode(60);
43     TreeNode* node6 = createNode(75);
44     TreeNode* node7 = createNode(47);
45     TreeNode* node8 = createNode(83);
46
47     // 创建二叉树
48     createTree(root, 89, 45, 97);
49     createTree(node2, 45, 71, 60);
50     createTree(node3, 97, 75, -1);
51     createTree(node4, 71, -1, -1);
52     createTree(node5, 60, 47, 83);
53
54     // 前序遍历
55     vector<int> preorderResult;
56     preorderTraversal(root, preorderResult);
57
58     // 输出结果
59     for (int i : preorderResult) {
60         cout << i << " ";
61     }
62     cout << endl;
63
64     return 0;
65 }
```

The output of the program is shown in the console:

```
中序遍历结果: 71 45 75 60 47 89 97 75
后序遍历结果: 71 75 47 60 45 75 97 89
PS C:\Users\JHERU\Desktop\class\Data_Structure\work5\build> . "C:\Users\JHERU\Desktop\class\Data_Structure\work5\build\work5.exe"
前序遍历结果: 89 45 71 60 75 47 97 75
中序遍历结果: 71 45 75 60 47 89 97 75
后序遍历结果: 71 75 47 60 45 75 97 89
PS C:\Users\JHERU\Desktop\class\Data_Structure\work5\build> . "C:\Users\JHERU\Desktop\class\Data_Structure\work5\build\work5.exe"
前序遍历结果: 89 45 71 60 47 83 97 75
中序遍历结果: 71 45 47 60 83 89 97 75
后序遍历结果: 71 47 83 60 45 75 97 89
PS C:\Users\JHERU\Desktop\class\Data_Structure\work5\build> .
```

CODE:

```

1 #include <iostream>
2 #include <vector>
3 // 树的结构体
4 struct TreeNode
5 {
6     int data; // 数据
7     TreeNode *left; // 左子节点
8     TreeNode *right; // 右子节点
9 };
10
11 /**
12  * @brief 创建一个二叉树节点
13  * @param data 节点数据
14  * @return 新创建的节点指针
15  */
16 TreeNode *createNode(int data)
17 {
18     TreeNode *newNode = new TreeNode();
19     if (newNode)
20     {
21         newNode->data = data;
22         newNode->left = newNode->right = nullptr;
23     }
24     return newNode;
25 }
26
27 /**
28  * @brief 创建节点之间的链接
29  * @param parent 父节点
30  * @param leftChild 左子节点
31  * @param rightChild 右子节点
32  */
33 void createLink(TreeNode *parent, TreeNode *leftChild, TreeNode *rightChild)
34 {
35     parent->left = leftChild;
36     parent->right = rightChild;
37 }
38
39 /**
40  * @brief 前序遍历二叉树
41  * @param root 根节点
42  * @param result 存储遍历结果的向量
43  */
44 void preorderTraversal(TreeNode *root, std::vector<int> &result)
45 {
46     if (root)
47     {
48         result.push_back(root->data);
49         preorderTraversal(root->left, result);
50         preorderTraversal(root->right, result);
51     }
52 }
53
54 /**
55  * @brief 中序遍历二叉树
56  * @param root 根节点
57  * @param result 存储遍历结果的向量
58  */
59 void inorderTraversal(TreeNode *root, std::vector<int> &result)
60 {
61     if (root)
62     {
63         inorderTraversal(root->left, result);
64         result.push_back(root->data);
65         inorderTraversal(root->right, result);
66     }
67 }
68
69 /**
70  * @brief 后序遍历二叉树
71  * @param root 根节点
72  * @param result 存储遍历结果的向量
73  */
74 void postorderTraversal(TreeNode *root, std::vector<int> &result)
75 {
76     if (root)
77     {
78         postorderTraversal(root->left, result);
79         postorderTraversal(root->right, result);
80         result.push_back(root->data);
81     }
82 }
83
84 int main()
85 {
86     // 创建二叉树的节点数据及对应关系
87     TreeNode *root = createNode(89);
88     TreeNode *node2 = createNode(45);
89     TreeNode *node3 = createNode(97);
90     TreeNode *node4 = createNode(71);
91     TreeNode *node5 = createNode(69);
92     TreeNode *node6 = createNode(73);
93     TreeNode *node7 = createNode(47);
94     TreeNode *node8 = createNode(83);
95
96     createLink(root, node2, node3);
97     createLink(node2, node4, node5);
98     createLink(node2, nullptr, node6);
99     createLink(node5, node7, node8);
100
101     // 前序遍历
102     std::vector<int> preorderResult;
103     preorderTraversal(root, preorderResult);
104     std::cout << "前序遍历结果: ";
105     for (int num : preorderResult)
106     {
107         std::cout << num << " ";
108     }
109     std::cout << std::endl;
110
111     // 中序遍历
112     std::vector<int> inorderResult;
113     inorderTraversal(root, inorderResult);
114     std::cout << "中序遍历结果: ";
115     for (int num : inorderResult)
116     {
117         std::cout << num << " ";
118     }
119     std::cout << std::endl;
120
121     // 后序遍历
122     std::vector<int> postorderResult;
123     postorderTraversal(root, postorderResult);
124     std::cout << "后序遍历结果: ";
125     for (int num : postorderResult)
126     {
127         std::cout << num << " ";
128     }
129     std::cout << std::endl;
130
131     return 0;
132 }
133
134
135
136

```

