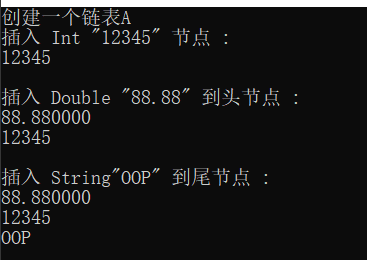
**用C++实现异质链表报告**

# 运行结果与截图

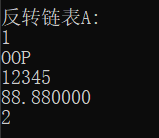
## 创建链表A并插入一个节点“12345”，插入 Double "88.88" 到头节点，插入 String"OOP" 到尾节点 :



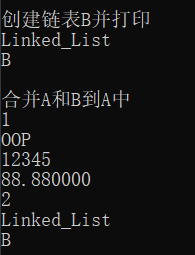
## 插入 2,1 到头节点 and 插入 1,2 到尾节点，删除头节点(1)，删除尾节点(2) :

## 

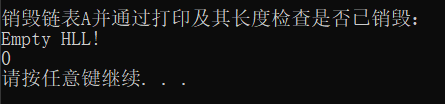
## 反转链表A



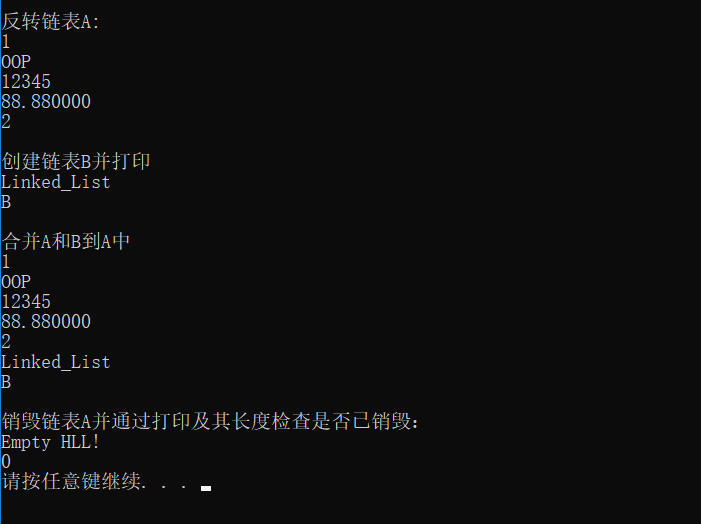
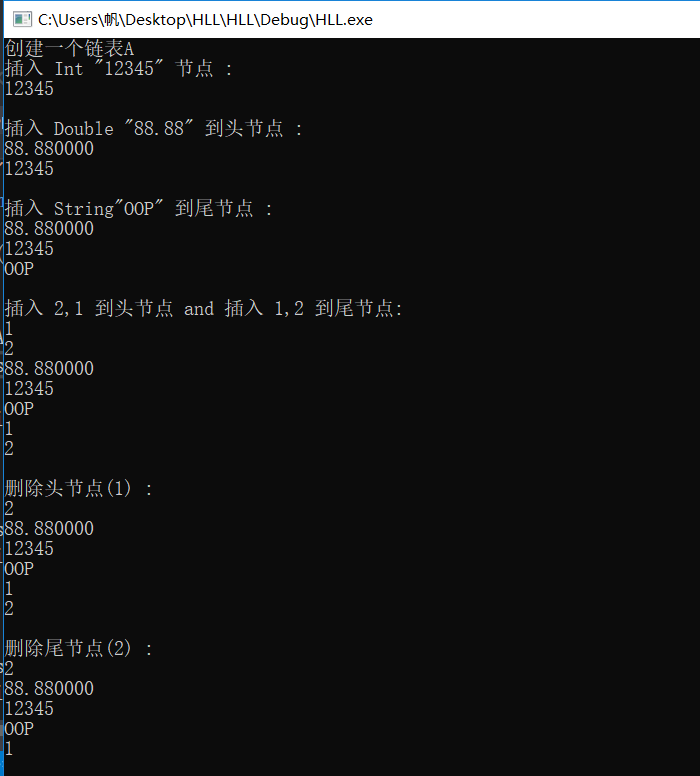
## 创建链表B插入”Linked\_List”、”B”，并且合并A与B到A中



## 销毁链表A并通过打印及其长度检查是否已销毁：

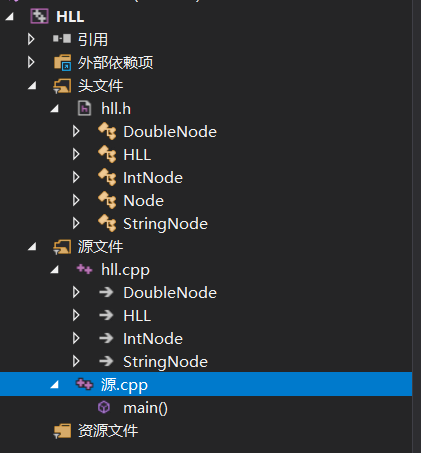


整体运行测试结果如下图：



# 具体代码

## 文件架构如下



## 异质链表头文件hll.h

1. #pragma once
2. #include<string>
3. #include<iostream>
4. #include<cstdio>
5. **using** **namespace** std;
7. **class** Node                                                  //基类
8. {
9. **public**:
10. Node() {}
11. **virtual** ~Node() {}                                      //虚析构函数
12. Node \*&next() { **return** Next; }                          //获取下一个指针
13. **virtual** **void** print() **const** = 0;                         //纯虚函数打印节点
14. **virtual** **void** union\_other(Node \*&a) **const** = 0;               //纯虚函数合并链表
15. **private**:
16. Node \*Next;                                             //指向下一节点
17. };
19. **class** IntNode :**public** Node                                  //整数继承类节点
20. {
21. **private**:
22. **int** Int\_value;
23. **public**:
24. IntNode(**int** a);
25. ~IntNode() {}
26. **virtual** **void** union\_other(Node \*&a) **const**;
27. **virtual** **void** print() **const**;
28. };
30. **class** StringNode :**public** Node                               //字符串继承类
31. {
32. **private**:
33. **char** \*Char\_value;
34. **public**:
35. StringNode(**const** **char** \*a);
36. ~StringNode();
37. **virtual** **void** union\_other(Node \*&a) **const**;
38. **virtual** **void** print() **const**;
39. };
41. **class** DoubleNode :**public** Node                               //双精度浮点数继承类
42. {
43. **private**:
44. **double** Double\_value;
45. **public**:
46. DoubleNode(**double** a);
47. ~DoubleNode() {}
48. **virtual** **void** union\_other(Node \*&a) **const**;
49. **virtual** **void** print() **const**;
50. };
52. //链表类
53. **class** HLL {
54. **public**:
55. Node \*Head;
56. Node \*Tail;
57. **int** Length;
58. HLL();
59. ~HLL();
60. **void** print()**const**; //打印所有节点
61. Node \* get\_head(); //返回链表的头指针
62. Node \* get\_tail(); //返回链表的尾指针
63. Node \* operator [](**int** index); //使用[]控制链接列表index是要操作的节点的位置
64. **const** **int** length()**const**; //返回链表的长度
65. **void** insert\_head(**int** a); //将IntNode插入链表的前面
66. **void** insert\_head(**double** a); //将DoubleNode插入链表的前面
67. **void** insert\_head(**const** **char** \* a); //将StringNode插入链接列表的前面
68. **void** insert\_tail(**int** a); //将IntNode插入链表的后面
69. **void** insert\_tail(**double** a); //将DoubleNode插入链表的后面
70. **void** insert\_tail(**const** **char** \* a); //将StringNode插入链表的后面
71. **void** delete\_head(); //删除链表的头节点
72. **void** delete\_tail(); //删除链表的尾节点
73. **void** reverse(); //反转链表
74. **void** destory(); //销毁链表
75. **void** union\_other(**const** HLL& a); //合并链表
76. };

## 异质链表具体操作函数定义文件hll.cpp

1. #include "hll.h"
3. HLL::HLL()
4. {
5. Head = Tail = NULL;                                             //initialize the list
6. Length = 0;
7. }
9. HLL::~HLL()
10. {
11. destory();
12. }
14. **void** HLL::print() **const**
15. {
16. Node \*Temp = Head;
17. **if** (!Temp)
18. cout << "Empty HLL!" << endl;
19. **else**
20. {
21. **while** (Temp)
22. {
23. Temp->print();                                           //call virtual function
24. Temp = Temp->next();
25. }
26. }
27. }
29. Node \* HLL::get\_head()
30. {
31. **return** Head;                                                    //return Head
32. }
34. Node \* HLL::get\_tail()
35. {
36. **return** Tail;                                                    //return Tail
37. }
39. Node \* HLL::operator[](**int** index)
40. {
41. **int** count = 0;
42. Node \*Temp = Head;
43. **while** (Temp)                                                        //find the index poision's node
44. {
45. **if** (count == index)
46. **break**;
47. Temp = Temp->next();
48. count++;
49. }
50. **return** Temp;                                                    //return the node's pointer
51. }
53. **const** **int** HLL::length() **const**
54. {
55. **return** Length;                                                  //return the length
56. }
58. **void** HLL::insert\_head(**int** a)                                        //insert Int node into head (function overload)
59. {
60. Node \*Temp = **new** IntNode(a);
61. Temp->next() = Head;
62. Head = Temp;
63. **if** (!Tail)
64. Tail = Temp;
65. Length++;
66. }
68. **void** HLL::insert\_head(**double** a)                                 //insert Double node into head (function overload)
69. {
70. Node \*Temp = **new** DoubleNode(a);
71. Temp->next() = Head;
72. Head = Temp;
73. **if** (!Tail)
74. Tail = Temp;
75. Length++;
76. }
78. **void** HLL::insert\_head(**const** **char** \* a)                                   //insert String node into head (function overload)
79. {
80. Node \*Temp = **new** StringNode(a);
81. Temp->next() = Head;
82. Head = Temp;
83. **if** (!Tail)
84. Tail = Temp;
85. Length++;
86. }
88. **void** HLL::insert\_tail(**int** a)                                        //insert Int node into tail(function overload)
89. {
90. Node \*Temp = **new** IntNode(a);
91. Temp->next() = NULL;
92. **if** (Tail)                                                       //find the tail node
93. Tail->next() = Temp;
94. Tail = Temp;
95. **if** (!Head)
96. Head = Temp;
97. Length++;
98. }
100. **void** HLL::insert\_tail(**double** a)                                 //insert Double node into head (function overload)
101. {
102. Node \*Temp = **new** DoubleNode(a);
103. Temp->next() = NULL;
104. **if** (Tail)
105. Tail->next() = Temp;
106. Tail = Temp;
107. **if** (!Head)
108. Head = Temp;
109. Length++;
110. }
112. **void** HLL::insert\_tail(**const** **char** \* a)                                   //insert String node into head (function overload)
113. {
114. Node \*Temp = **new** StringNode(a);
115. Temp->next() = NULL;
116. **if** (Tail)
117. Tail->next() = Temp;
118. Tail = Temp;
119. **if** (!Head)
120. Head = Temp;
121. Length++;
122. }

125. **void** HLL::delete\_head()                                         //Delete the head node (function overload)
126. {
127. **if** (!Head)
128. {
129. cout << "The list is empty!" << endl;
130. **return**;
131. }
132. Node \*Temp = Head;
133. Head = Temp->next();
134. **delete** Temp;
135. Length--;
136. }
138. **void** HLL::delete\_tail()                                         //Delete the tail node (function overload)
139. {
140. **if** (!Tail)
141. {
142. cout << "The list is empty!" << endl;
143. **return**;
144. }
145. Node \*Temp = Head;
146. **if** (1 == Length)
147. {
148. **delete** Tail;
149. Head = Tail = NULL;
150. }
151. **else**
152. {
153. **while** (Temp->next()->next())
154. Temp = Temp->next();
155. **delete** Temp->next();
156. Temp->next() = NULL;
157. Tail = Temp;
158. }
159. Length--;
160. }

163. **void** HLL::reverse()                                             //Reverse the linked list
164. {
165. **if** (1 >= Length)
166. **return**;
167. Node \*FTemp = Head;
168. Node \*NTemp = Head->next();
169. **while** (NTemp)
170. {
171. FTemp->next() = NTemp->next();
172. NTemp->next() = Head;
173. Head = NTemp;
174. NTemp = FTemp->next();
175. }
176. Tail = FTemp;
177. }
179. **void** HLL::destory()                                             //Destory the linked list
180. {
181. Node \*Temp = Head;
182. Node \*T;
183. **while** (Temp)
184. {
185. T = Temp;
186. Temp = Temp->next();
187. **delete** T;
188. }
189. Head = Tail = NULL;
190. Length = 0;
191. }
193. **void** HLL::union\_other(**const** HLL& a)                                 //Union two linked list
194. {
195. Node \*TTemp = Tail;
196. Node \*FTemp = a.Head;
197. **while** (FTemp)
198. {
199. FTemp->union\_other(TTemp);
200. FTemp = FTemp->next();
201. TTemp = TTemp->next();
202. }
203. }

206. IntNode::IntNode(**int** a)                                             //IntNode class
207. {
208. Int\_value = a;
209. }
211. **void** IntNode::union\_other(Node \*& a) **const**                              //IntNode union
212. {
213. a->next() = **new** IntNode(Int\_value);
214. a->next()->next() = NULL;
215. }
217. **void** IntNode::print() **const**                                         //IntNode print
218. {
219. cout << Int\_value << endl;
220. }
222. DoubleNode::DoubleNode(**double** a)                                    //DoubleNode class
223. {
224. Double\_value = a;
225. }
227. **void** DoubleNode::union\_other(Node \*& a) **const**                           //DoubleNode union
228. {
229. a->next() = **new** DoubleNode(Double\_value);
230. a->next()->next() = NULL;
231. }
233. **void** DoubleNode::print() **const**                                      //Double print
234. {
235. printf("%lf\n", Double\_value);
236. }
238. StringNode::StringNode(**const** **char** \* a)                                  //StringNode class
239. {
240. Char\_value = **new** **char**[strlen(a) + 1];
241. strcpy(Char\_value, a);
242. }
244. StringNode::~StringNode()                                           //virtual StringNode destructor function to release the space applicated in heap
245. {
246. **delete** Char\_value;
247. }
249. **void** StringNode::union\_other(Node \*& a) **const**                           //StringNode union
250. {
251. a->next() = **new** StringNode(Char\_value);
252. a->next()->next() = NULL;
253. }
255. **void** StringNode::print() **const**                                      //StringNode print
256. {
257. cout << Char\_value << endl;
258. }

## 测试代码test.cpp(即包含main()来测试的cpp)

1. #include"hll.h"
2. **using** **namespace** std;
4. **int** main()
5. {
6. cout << "创建一个链表A" << endl;
7. HLL A;
8. A.insert\_head(12345);
9. cout << "插入 Int \"12345\" 节点 : " << endl;
10. A.print();
11. cout << endl;
13. cout << "插入 Double \"88.88\" 到头节点 : " << endl;
14. A.insert\_head(88.88);
15. A.print();
16. cout << endl;
18. cout << "插入 String\"OOP\" 到尾节点 : " << endl;
19. A.insert\_tail("OOP");
20. A.print();
21. cout << endl;
23. cout << "插入 2,1 到头节点 and 插入 1,2 到尾节点: " << endl;
24. A.insert\_head(2);
25. A.insert\_head(1);
26. A.insert\_tail(1);
27. A.insert\_tail(2);
28. A.print();
29. cout << endl;
31. cout << "删除头节点(1) : " << endl;
32. A.delete\_head();
33. A.print();
34. cout << endl;
36. cout << "删除尾节点(2) : " << endl;
37. A.delete\_tail();
38. A.print();
39. cout << endl;
41. cout << "反转链表A: " << endl;
42. A.reverse();
43. A.print();
44. cout << endl;
46. cout << "创建链表B并打印 " << endl;
47. HLL B;
48. B.insert\_head("Linked\_List");
49. B.insert\_tail("B");
50. B.print();
51. cout << endl;
53. cout << "合并A和B到A中" << endl;
54. A.union\_other(B);
55. A.print();
56. cout << endl;
58. cout << "销毁链表A并通过打印及其长度检查是否已销毁：" << endl;
59. A.destory();
60. A.print();
61. cout << A.length() << endl;
62. system("pause");
63. **return** 0;
64. }