```cpp
#include <iostream>
#include <iomanip>

using namespace std;


struct Node
{
    int data;
    Node* next;
    Node* prev;
};

class DoublyLinkedList
{
public:
    void push(int data);
    void pop();
    int Size();
    void const print();
    int& at(int idx);
    void insert(int data, int pos);
    void remove(int pos);
    DoublyLinkedList()
    {
        data = 0;
        idx = 0;
        pos = 0;
        head = nullptr;
        tail = nullptr;
        size = 0;
    }
    ~DoublyLinkedList() //destructor is responsible for freeing the memory that was allocated for the nodes in the list when an objectw was destroyed
    {
        Node* current = head;
        while (current != NULL)
        {
            Node* next = current->next;
            delete current;
            current = next;
        }
        head = NULL;
        tail = NULL;
        size = 0;
    }
    DoublyLinkedList(const DoublyLinkedList& other) //creates a new empty list and then copies all the elements from the other list using the
```

```
        push function
47      {
48          head = NULL;
49          tail = NULL;
50          size = 0;
51          Node* current = other.head;
52          while (current != NULL)
53          {
54              push(current->data);
55              current = current->next;
56          }
57      }
58      DoublyLinkedList& operator=(const DoublyLinkedList& other) //The copy
          assignment operator first checks if the lists are the same object as
           the other,
59      {// If same, function does nothing and returns *this. If different,
          clears the current list and copies all the elements from the other
          list using the push
60          if (this != &other)
61          {
62              while (head != NULL) {
63                  Node* nodeToRemove = head;
64                  head = head->next;
65                  delete nodeToRemove;
66              }
67              tail = NULL;
68              size = 0;
69              Node* current = other.head;
70              while (current != NULL) {
71                  push(current->data);
72                  current = current->next;
73              }
74          }
75          return *this;
76      }
77
78  private:
79      int data;
80      int idx;
81      int pos;
82      Node* head;
83      Node* tail;
84      int size;
85  };
86
87  void DoublyLinkedList::push(int data)
88  {
89      Node* newNode = new Node;
90      newNode->data = data;
```

```
 91      newNode->next = NULL;
 92      newNode->prev = tail;
 93      if (tail != NULL) {
 94          tail->next = newNode;
 95      }
 96      tail = newNode;
 97      if (head == NULL) {
 98          head = newNode;
 99      }
100      size++;
101  }
102
103
104  void DoublyLinkedList::pop()
105  {
106      Node* nodeToRemove = tail;
107      int value = nodeToRemove->data;
108      tail = tail->prev;
109      if (tail != NULL) {
110          tail->next = NULL;
111      }
112      else {
113          head = NULL;
114      }
115  }
116
117  int DoublyLinkedList::Size()
118  {
119      return size;
120  }
121
122  void const DoublyLinkedList::print()
123  {
124      Node* current = head;
125      while (current != NULL) {
126          cout << current->data << " ";
127          current = current->next;
128      }
129      cout << endl;
130  }
131
132
133  int& DoublyLinkedList::at(int idx)
134  {
135      if (idx < 0 || idx >= size)
136      {
137          cout << "Invalid index." << endl;
138          exit(1);
139      }
```

```
140     Node* current = head;
141     for (int i = 0; i < idx; i++) {
142         current = current->next;
143     }
144     return current->data;
145 }
146 
147 void DoublyLinkedList::insert(int data, int pos)
148 {
149     if(pos < 0 || pos > size)
150     {
151         cout << "invalid insert" << endl;
152     }
153     else
154     {
155         Node* newNode = new Node;
156         newNode->data = data;
157         if (pos == 0) {
158             newNode->next = head;
159             newNode->prev = NULL;
160             if (head != NULL) {
161                 head->prev = newNode;
162             }
163             head = newNode;
164             if (tail == NULL) {
165                 tail = newNode;
166             }
167         }
168         else {
169             Node* prevNode = head;
170             for (int i = 0; i < pos - 1; i++) {
171                 prevNode = prevNode->next;
172             }
173             newNode->next = prevNode->next;
174             newNode->prev = prevNode;
175             if (prevNode->next != NULL) {
176                 prevNode->next->prev = newNode;
177             }
178             else {
179                 tail = newNode;
180             }
181             //prevNode->next = newNode;
182         }
183         size++;
184     }
185 }
186 
187 
188 
```

```
189 void DoublyLinkedList::remove(int pos)
190 {
191     if (pos < 0 || pos >= size) {
192         cout << "invalid removal"<<endl;
193         throw out_of_range("Invalid position");
194     }
195     else if (pos == 0)
196     {
197         Node* nodeToRemove = head;
198         int value = nodeToRemove->data;
199         head = nodeToRemove->next;
200         if (head != NULL) {
201             head->prev = NULL;
202         }
203         else {
204             tail = NULL;
205         }
206         delete nodeToRemove;
207         size--;
208     }
209     else
210     {
211         Node* nodeToRemove = head;
212         if (pos == 0) {
213             head = nodeToRemove->next;
214             if (head != NULL) {
215                 head->prev = NULL;
216             }
217             else {
218                 tail = NULL;
219             }
220         }
221         else if (pos == size - 1) {
222             nodeToRemove = tail;
223             tail = nodeToRemove->prev;
224             tail->next = NULL;
225         }
226         else {
227             for (int i = 0; i < pos; i++) {
228                 nodeToRemove = nodeToRemove->next;
229             }
230             nodeToRemove->prev->next = nodeToRemove->next;
231             nodeToRemove->next->prev = nodeToRemove->prev;
232         }
233         delete nodeToRemove;
234         size--;
235     }
236 }
237
```

```
238
239 void test() {
240     DoublyLinkedList dll;
241
242     dll.push(1);
243     dll.push(2);
244     dll.push(3);
245
246     DoublyLinkedList dllCopy = dll;
247
248     dllCopy.push(4);
249     dllCopy.remove(1);
250     dllCopy.remove(0);
251     dllCopy.print();
252     dllCopy.insert(1, 3);
253
254     dll.print();
255     dllCopy.print();
256
257     dll = dllCopy;
258     dll.print();
259 }
260
261 int main()
262 {
263     test();
264
265     return 0;
266 }
267
268
```