```
#ifndef _LINKEDLIST_H_
#define _LINKEDLIST_H_
using namespace std;

// Linked List Template
template <typename T>
class linkedlist {
private:
    class node
    {
        friend class linkedlist<T>;

        private:
            T data;
            node* next;

        public:
            node(T d, node* n= NULL): data(d), next(n) {}
    };

    node* head;
    node* tail;

public:
    int count;
    // Default Constructor
    linkedlist(void): head(NULL), tail(NULL), count(0) {}
    // Copy Constructor
    linkedlist(const linkedlist<T>& src);
    // Destructor
    ~linkedlist(void);
    int size(void) { return count;}
    bool empty(void) {return count==0 ;}
    void push_back(T);
    void push_front(T);
    void pop_front(void);
    void pop_back(void);
    void display(void);

};

// Insert elements from behind
template <typename T>
void linkedlist<T>::push_back(T d)
{
    node* temp = new node(d, NULL);
    if(this->empty())
    {
        head = temp;
    }
    else
    {
        tail->next = temp;
    }

    tail = temp;
    count++;
}

// Insert elements from front
template<typename T>
```

```
62  void linkedlist<T>::push_front(T d)
63  {
64      node* temp = new node(d, head); // new node linked to head.
65      if(this->empty())
66      {
67          head = temp;
68          tail = temp;
69      }
70      else
71      {
72          head = temp;
73      }
74      count++;
75  }
76
77  // Delete elements from front
78  template<typename T>
79  void linkedlist<T>::pop_front(void)
80  {
81      if(head==NULL)
82          cout << "Underflow\n";
83      //If there is only one node, then set head and tail to NULL
84      if(this->size()==1)
85      {
86          head=NULL;
87          tail=NULL;
88          count--;
89
90      }
91      else
92      {
93          node* oldhead = head;
94          delete oldhead;
95          head = head->next;
96          count--;
97      }
98  }
99
100 // Delete elements from behind
101 template<typename T>
102 void linkedlist<T>::pop_back(void)
103 {
104     if(tail==NULL)
105         cout << "Underflow\n";
106
107     //If there is only one node, then set head and tail to NULL
108     if(this->size()==1)
109     {
110         head=NULL;
111         tail=NULL;
112         count--;
113     }
114     else
115     {
116         node* itr = head;
117         //find the node prior to tail node
118         while(itr->next!=tail)
119         {
120             itr = itr->next;
121         }
122         node* oldtail = itr->next;
123         delete oldtail;
124         itr->next = NULL;
```

```
125         tail = itr;
126         count--;
127     }
128 
129 }
130 
131 // Display Function
132 template <typename T>
133 void linkedlist<T>::display(void)
134 {
135     node* current = head;
136     if(current!=NULL)
137     {
138         cout << this << ": ";
139         cout << "[ ";
140     }
141     if(current!=NULL)
142     {
143         while(current->next!=NULL)
144         {
145             cout << current->data << " ]->[ ";
146             current = current->next;
147         }
148         cout << current->data;
149     }
150     else
151     {
152         cout << "Underflow!\n";
153         return;
154     }
155 
156     cout << " ]->NULL\n";
157 }
158 
159 // Copy Constructor
160 template <typename T>
161 linkedlist<T>::linkedlist(const linkedlist<T>& oldlist): head(NULL), tail(NULL),
    count(0)
162 {
163     node* current = oldlist.head;
164     while(current!=NULL)
165     {
166         this->push_back(current->data);
167         current = current->next;
168     }
169 }
170 
171 // Destructor
172 template <typename T>
173 linkedlist<T>::~linkedlist(void)
174 {
175 
176     while(!this->empty())
177     {
178         this->pop_front();
179     }
180 }
181 #endif
```