Linked Lists – Explanatory Notes

A linked list is an ordered list containing data in non-contiguous memory locations. Each "node" of the linked list contains the data (which is private to the node), and is created on the free store (heap) as and when new data is to be added to the list. Along with the data, the node also contains a field to store the address of the next node, since the nodes are not located in contiguous memory locations. This pointer is named `link`.

The "manager" of the linked list is `Chain`, which contains a pointer to store the start of the list (`first`), as well as member functions to manipulate the list – in terms of inserting a node to the list, deleting a node, and print all members of the list. In order to enable `Chain` to access the privately stored data of `ChainNode`, the `ChainNode` declares `Chain` as its "friend" (line 6).

Constructors and destructors are special methods of classes which are used automatically when objects are created (instantiated) and the program ends, respectively. What should be done when a node is created? Its fields should be initialized. That's exactly what happens in the constructor of `ChainNode` – as shown in lines 8 and 9, which constitute the constructor of the class. What should happen before the program ends? The nodes which were created in the heap should be freed up. And this is what a destructor accomplishes – in lines 24 through 31.

Note that member functions can also be defined outside the class. The class simply has to contain the declarations of those member functions (e.g., lines 20-23).

```
 1    #include <iostream>
 2
 3    using namespace std;
 4
 5    class ChainNode {
 6      friend class Chain;
 7      public:
 8          ChainNode (int element=0, ChainNode *next=0) // constructor
 9          {     data = element; link = next;     }
10      private:
11          int data;
12          ChainNode *link;
13    };
14
15    class Chain {
16      private:
17          ChainNode *first;
18      public:
19          Chain() { first = NULL; } // constructor
20          void insertbegin(int);
21          void insertend(int);
22          void remove(int);
23          void printchain();
24          ~Chain() {            // destructor
25                     ChainNode *tmp;
26                     while (first) {
27                          tmp = first->link;
28                          delete first;
29                          first = tmp;
30                          }
31                }
32
33    };
```

```
34
35    int main()
36    {
37      Chain c;
38      short int choice;
39      int data;
40      do
41      {
42        cout << endl << " Insert at: 1. Beginning 2. End 3. Stop " << endl;
43        cout << "   Enter your choice (1/2/3): ";
44        cin >> choice;
45        switch (choice)
46        {
47          case 1: cout << "Data to be inserted at beginning: ";
48                cin >> data;
49                c.insertbegin(data);
50                break;
51          case 2: cout << "Data to be inserted at end: ";
52                cin >> data;
53                c.insertend(data);
54                break;
55        }
56      } while (choice != 3);
57      c.printchain();
58      cout << "Enter node to be removed : ";
59      cin >> data;
60      c.remove(data);
61      c.printchain();
62      return 0;
63    }
64
65    void Chain::insertbegin(int x)
66    {
67      if (first)
68      {
69          // *** your code goes here... ***
70          // *** no more than 3-4 lines! ****
71      }
72      else
73      {
74          // *** your code goes here... ***
75          // *** no more than 1-2 lines! ****
76      }
77    }
78
79    void Chain::insertend(int x)
80    {
81      if (first)  // list already exists
82      {
83          ChainNode *tmp=first;
84          while (tmp->link != 0)     // reach the last node
85                tmp = tmp->link;
86          tmp->link = new ChainNode(x,0);  // add new one
87      }
88      else // empty list; creating the first node of the list
89          first = new ChainNode(x,0);
90    }
```

```
 91
 92     void Chain::remove(int x)
 93     {
 94       ChainNode *tmp = first;
 95       while (first && first->data == x)
 96       {
 97           first = first->link;
 98           delete tmp;
 99           tmp = first;
100       }
101       if (first == nullptr) return;
102       ChainNode *prev = first;
103       while (tmp)
104       {
105           if (tmp->data == x)
106           {
107             prev->link = tmp->link;
108             delete tmp;
109             tmp = prev->link;
110             continue;
111           }
112           prev = tmp;
113           tmp = tmp->link;
114       }
115     }
116
117     void Chain::printchain()
118     {
119       ChainNode *tmp = first;
120       while (tmp)
121       {
122           cout << tmp->data << endl;
123           tmp = tmp->link;
124       }
125     }
```

As long as the address is a valid one (which is a non-zero value), print the data of the node.
After printing, advance to the next node.