1. **Linked List Class Template**

You created a linked list class for manipulating a linked list of integers. What if you want a linked list of Temperatures? Using the previous method, you would have to create another linked list class. *Template* allows you to create a linked list (or any other class) for a generic data type, and then the user simply specifies the actual data type.

The linked list template header file includes both the declaration and the definitions of the member functions. There cannot be a separate .cpp file for defining the member functions. A template prefix, such as

```
template<class T>
```

is placed before the class declaration. The user identifier T in the above statement is a data type parameter. In the class definition, you would use T as the type rather than an actual type.

```cpp
#include <iostream>
using namespace std;

template <class T>
class LinkedList{
private:
    struct ListNode {
        T data;
        ListNode *next;
    };

    ListNode *head;
public:
    LinkedList();
    void InsertNode(T d);
    void PrintList();
    int DeleteNode(T d);
};

template <class T>
LinkedList<T>::LinkedList(){
    head = NULL;
}
```
template <class T>
void LinkedList<T>::InsertNode(T d) {
    ListNode *ptr = new ListNode;  // allocate memory for new node
    ptr->data = d;   // put data in new node
    if(head == NULL){   // if list is empty
        head = ptr;    // set head to point to this new node
        ptr->next = NULL;   // this is the end of the list
    } else{      // otherwise
        ptr->next = head; // insert new node at beginning of the list
        head = ptr;
    }
}

template <class T>
void LinkedList<T>::PrintList() {
    ListNode *ptr = head;
    cout << "Here's the linked list..." << endl;
    while(ptr != NULL){
        // in order for the cout to work you need to
        // overload operator << for your class T
        cout << ptr->data << endl;
        ptr = ptr->next;
    }
}

template <class T>
int LinkedList<T>::DeleteNode(T d) {
    ListNode *ptr = head;
    ListNode *previous; // trailing pointer
    // need to treat the first node differently
    // in order for the if condition to work you need to
    // overload operator == for your class T
    if(head->data == d){ // is the first node the one to delete?
        // Executing the above line will sometime cause the program to crash.
        // Do you know under what circumstance?
        ptr = head;
        head = head->next;
        delete ptr;
        return 1;
    } else{   // no, then traverse list to find node to delete
        // in order for the while condition to work you need to
        // overload operator != for your class T
        while((ptr != NULL) && (ptr->data != d)){
            previous = ptr;  // remember the previous node in the list
            ptr = ptr->next;  // next node in list
        }
        if(ptr){     // same as if(ptr != NULL)
            previous->next = ptr->next; // adjust pointer of previous node
            delete ptr;    // release memory occupied by node
            return 1;
        } else
            return 0;
    }
}
The main program...

```cpp
/*
 * This program demonstrates a template for a linked list
 * Created by Enoch Hwang on 2/1/10.
 * Copyright 2010 La Sierra University. All rights reserved.
 */

#include <iostream>
#include "Linked List.h"
using namespace std;

int main (int argc, char * const argv[]) {
    LinkedList<int> L1;   // create a linked list of integers
    LinkedList<float> L2;  // create a linked list of floats
    LinkedList<char> L3;  // create a linked list of chars

    cout << "Inserting node 6 to L1..." << endl;
    L1.InsertNode(6);
    cout << "Inserting node 6.4 to L2..." << endl;
    L2.InsertNode(6.4);
    cout << "Inserting node 'C' to L3..." << endl;
    L3.InsertNode('C');

    L1.PrintList();
    cout << "Deleting node 6 from L1..." << endl;
    if(L1.DeleteNode(6))
        cout << "  Node deleted\n";
    else
        cout << "  Node not deleted\n";
    L1.PrintList();

    return 0;
}
```

Sample output

Inserting node 6 to L1...
Inserting node 6.4 to L2...
Inserting node 'C' to L3...
Here's the linked list...
6
Deleting node 6 from L1...
Node deleted
Here's the linked list...

2. Exercise

1. Create a template linked list of Temperatures.