A stack will be a list of nodes that are linked together.

The stack has methods push: to insert a node at the top, and pop: to remove the top node and return its value.

ListNode.h *there is no ListNode.cpp*

```
//Template ListNode class definition.
#ifndef LISTNODE_H
#define LISTNODE_H

template <typename NODETYPE> class Stack;
template <typename NODETYPE>
class ListNode
{
    friend class Stack <NODETYPE>; //make Stack a friend
    public:
         ListNode(const NODETYPE &); //constructor
         NODETYPE getData()const; //return data in node
         NODETYPE data; //data
         ListNode <NODETYPE> *nextPtr; //next node in list
 };//end class ListNode

//constructor
template <typename NODETYPE>
ListNode <NODETYPE>::ListNode(const NODETYPE &info )
       : data( info ), nextPtr(NULL)
 {
       //empty body
 }//end ListNode constructor

//return copy of data in node
 template <typename NODETYPE>
 NODETYPE ListNode<NODETYPE>::getData()const
 {
       return data;
 }//end function getData
#endif
```

Stack.h *there is no Stack.cpp*

```
//Template Stack class definition.
#ifndef STACK_H
#define STACK_H
#include<iostream>
#include"ListNode.h"//ListNode class definition
using namespace std;
template<typename NODETYPE>
class Stack {
    public:
       Stack();//constructor
       ~Stack();//destructor
    void push(const NODETYPE &);
    NODETYPE pop();
       bool isEmpty() const;
       void print() const;

   private:
     ListNode<NODETYPE>*firstPtr;//pointer to first node
     //utility function to allocate new node
     ListNode<NODETYPE>*getNewNode(const NODETYPE &);
};//end class Stack

  //default constructor
 template<typename NODETYPE>
 Stack<NODETYPE>::Stack():firstPtr(NULL ) {
       //empty body
 } //end Stack constructor
 //destructor
  template<typename NODETYPE>
  Stack<NODETYPE>::~Stack() {
       if( !isEmpty() ) {  //List is not empty
           cout<<"Destroying nodes ...\n";
           ListNode<NODETYPE>*currentPtr=firstPtr;
           ListNode<NODETYPE>*tempPtr;
           while(currentPtr !=0) { //delete remaining nodes
             tempPtr=currentPtr;
             cout<<tempPtr->getData()<< '\n';
             currentPtr=currentPtr->nextPtr;
             delete tempPtr;
           }//end while
       }//end if
       cout<<"All nodes destroyed\n\n";
  }//end List destructor
// insert node at front of list: push
  template <typename NODETYPE>
  void Stack<NODETYPE>::push(const NODETYPE &value ) {
       ListNode<NODETYPE>*newPtr=getNewNode(value );//new node
       if( isEmpty() )//List is empty
         firstPtr=newPtr;//new list has only one node
       else  { //List is not empty
         newPtr->nextPtr=firstPtr;//point new node to previous 1st node
         firstPtr=newPtr;
       } //end else
  } //end insertAtFront
```

```
//delete node from front of list: pop
  template<typename NODETYPE>
  NODETYPE Stack<NODETYPE>::pop()  {
         if( isEmpty() )//Stack isempty
           return NULL;//pop unsuccessful
         else {
           ListNode<NODETYPE>*tempPtr=firstPtr;//hold temp Ptr to delete
           firstPtr=firstPtr->nextPtr;//point to previous 2nd node
           NODETYPE value=tempPtr->data;//return data being removed
           delete tempPtr;//reclaim p revious front node
           return value;//delete successful
         }//end else
  }//end function pop

  // isEmpty?
  template<typename NODETYPE>
  bool Stack<NODETYPE>::isEmpty()const {
       return firstPtr==NULL;
  }//end function isEmpty

  //return pointer to newly allocated node
    template<typename NODETYPE>
    ListNode<NODETYPE>*Stack<NODETYPE>::getNewNode(const NODETYPE &value )  {
         return new ListNode<NODETYPE>(value );
    }//end function getNewNode

  //displaycontentsofList
  template<typename NODETYPE>
  void Stack<NODETYPE>::print()const {
       if( isEmpty() ) { //List isempty
         cout<<"The list is empty\n\n";
       }//end if
       else {
         ListNode<NODETYPE>*currentPtr=firstPtr;
         cout<<"The list is: ";
         while(currentPtr !=NULL) { //get element data
           cout<<currentPtr->data<< ' ';
           currentPtr=currentPtr->nextPtr;
         }//endwhile
         cout<<"\n\n";
       }
 }//end function print
#endif
```

Main program to test the stack:

```
#include<iostream>
#include<string>
#include "Stack.h"
using namespace std;
//display program instructions to user
void instructions() {
       cout<<"Enter one of the following:\n"
       <<" 1 to push\n"
       <<" 2 to pop\n"
       <<" 3 to end processing\n";
} //end function instructions

 //function to test a Stack
 template<typename T>
 void testList(Stack<T> &listObject, const string &typeName )
  {
       cout<<"Testing a stack of "<<typeName<<" values\n";
       instructions();//display instructions

       int choice;//store user choice
       T value;//store input value

       do//perform user-selected actions
       {
         cout<<"?";
         cin>>choice;

        switch(choice ) {
          case 1:// push
           cout<<"Enter "<<typeName<<": ";
           cin>>value;
           listObject.push(value);
           listObject.print();
           break;
         case 2:// pop
              if(listObject.isEmpty()) cout<<"Can't pop, the list is empty\n";
              else cout<<listObject.pop()<<" popped\n";
       listObject.print();
              break;
   }//end switch
} while(choice<3 );//end do...while
} //end testList
void main() {
       cout<<"hello\n";
       //test Stack of string values
       Stack<string> myStack;
       testList( myStack, "string" );
       system("pause");
}
```