```
//
//  Display 17.18 Program Using the Stack Template Class
//  stack.cpp
//  AbsoluteCpp_ch17_17
//

//Program to demonstrate use of the Stack template class. 2 #include
 <iostream>
#include "stack.h"
#include "stack.cpp"
using std::cin;
using std::cout;
using std::endl;
using StackSavitch::Stack;

int main( )
{
    char next, ans;
    do
    {
        Stack<char> s;
        cout << "Enter a line of text:\n";
        cin.get(next);
        while (next != '\n')
        {
            s.push(next);
            cin.get(next);
        }

        cout << "Written backward that is:\n";
        while (! s.isEmpty() )
            cout << s.pop( );
        cout << endl;


        cout << "Again?(y/n): ";
        cin >> ans;
        cin.ignore(10000, '\n');
    }while (ans != 'n' && ans != 'N');
    return 0;
}
```

```
//
//  Display 17.18 Program Using the Stack Template Class
//  stack.cpp
//  AbsoluteCpp_ch17_17
//

//Program to demonstrate use of the Stack template class. 2 #include
 <iostream>
#include "stack.h"
#include "stack.cpp"
using std::cin;
using std::cout;
using std::endl;
using StackSavitch::Stack;

int main( )
{
    char next, ans;
    do
    {
        Stack<char> s;
        cout << "Enter a line of text:\n";
        cin.get(next);
        while (next != '\n')
        {
            s.push(next);
            cin.get(next);
        }

        cout << "Written backward that is:\n";
        while (! s.isEmpty() )
            cout << s.pop( );
        cout << endl;


        cout << "Again?(y/n): ";
        cin >> ans;
        cin.ignore(10000, '\n');
    }while (ans != 'n' && ans != 'N');
    return 0;
}
```

```
//
//  Display 17.17 Interface File for a Stack Template Class
//  stack.hpp
//  AbsoluteCpp_ch17_17
//

//This is the header file stack.h. This is the interface for the class
 //Stack, which is a template class for a stack of items of type T.
#ifndef STACK_H
#define STACK_H

namespace StackSavitch {
template<class T>
class Node
{
public:
    Node(T theData, Node<T>* theLink) : data(theData), link(theLink){}
    Node<T>* getLink( ) const { return link; }
    const T getData( ) const { return data; }
    void setData(const T& theData) { data = theData; }
    void setLink(Node<T>* pointer) { link = pointer; }
private:
    T data;
    Node<T> *link;
};

template<class T>
class Stack
{
public:
    Stack( );
    //Initializes the object to an empty stack.

    Stack(const Stack<T>& aStack);

    Stack<T>& operator =(const Stack<T>& rightSide);

    virtual ~Stack();

    void push(T stackFrame);
    //Postcondition: stackFrame has been added to the stack.

    T pop( );
    //Precondition: The stack is not empty.
    //Returns the top stack frame and removes that top
    //stack frame from the stack.
    bool isEmpty( ) const;
    //Returns true if the stack is empty. Returns false otherwise.
private:
    Node<T> *top;
};

} //StackSavitch

#endif //STACK_H
```

```
//
//  Display 17.19 Implementation of the Stack Template Class
//  stack.cpp
//  AbsoluteCpp_ch17_17
//

//This is the implementation file stack.cpp.
//This is the implementation of the template class Stack.
//The interface for the template class Stack is in the header file
//stack.h

#include <iostream>
#include <cstdlib>
#include <cstddef>
#include "stack.h"
using std::cout;

namespace StackSavitch
{

// Uses cstddef:
template<class T>
Stack<T>::Stack():top(NULL)
{
    //Intentionally empty
}

template<class T>
Stack<T>::Stack(const Stack<T>& aStack){
    // do your selft
}

template<class T>
Stack<T>& Stack<T>::operator =(const Stack<T>& rightSide){
    // do your selft
}

template<class T>
Stack<T>::~Stack( )
{
    T next;
    while (! isEmpty( ))
        next = pop( );//pop calls delete.
}

//Uses cstddef:
template<class T>
bool Stack<T>::isEmpty( ) const
{
    return (top == NULL);
}
template<class T>
void Stack<T>::push(T stackFrame){
    // do your selft
}
```

```
//Uses cstdlib and iostream:
template<class T>
T Stack<T>::pop( )
{
    if (isEmpty( ))
    {
        cout << "Error: popping an empty stack.\n";
        exit(1);
    }

    T result = top->getData( );
    Node<T> *discard;
    discard = top;
    top = top->getLink( );
    delete discard;
    return result;
}
}//StackSavitch
```