```
struct StackException {                          //Stack Exception
    StackException(char* m)
    {   cout << endl;
        cout << "I am the Stack and I am " << m << ".";
        cout << endl;
    }
};
template <class T>
class Stack {
    struct Element {
        Element* prev;                           //Points to previous element
        T value;                                 //Value in this Element
        Element(Element* p,T v):prev(p),value(v){}     //Constructor
    };
    Element* tos;                                    //Points to the Element on top of
the Stack
    void Kill(Element* e)                            //Reclaims storage from elements
    {   while (e) {                              //remaining on the Stack
            Element* p=e;
            e=e->prev;
            delete p;
        }
    }
public:
    Stack():tos(0){}                             //Constructor
    ~Stack(){Kill(tos);}                         //Destructor, reclaim storage
    bool IsFull(void){return false;}                 //Is the Stack full?
    bool IsEmpty(void){return tos==0;}               //Is the Stack empty?
    void Push(T v)                               //Put value v on top of the Stack
    {   if (IsFull()) throw StackException("full");
        Element* n=new Element(tos,v);
        tos=n;
    }
    T Pop(void)                                  //Remove and return the value on top
    of
    {   if (IsEmpty()) throw StackException("empty");   //the Stack
        Element* p=tos;
        T v=p->value;
        tos=p->prev;
        delete p;
        return v;
    }
};
```

Figure 1. template class *Stack*.

Example: Program **p01** reverses the characters of string "tilt" illustrating the last-in-first-out character of a stack.

```
#ifndef Stack_h
#define Stack_h 1
struct StackException {                          //Stack Exception
    StackException(char* m)
    {   cout << endl;
        cout << "I am the Stack and I am " << m << ".";
        cout << endl;
    }
};
template <class T>
class Stack {
    struct Element {
        Element* prev;                           //Points to previous element
        T value;                                 //Value in this Element
        Element(Element* p,T v):prev(p),value(v){}     //Constructor
    };
    Element* tos;                                    //Points to the Element on top of
the Stack
    void Kill(Element* e)                            //Reclaims storage from elements
    {   while (e) {                              //remaining on the Stack
            Element* p=e;
            e=e->prev;
            delete p;
        }
    }
public:
    Stack():tos(0){}                             //Constructor
    ~Stack(){Kill(tos);}                         //Destructor, reclaim storage
    bool IsFull(void){return false;}             //Is the Stack full?
    bool IsEmpty(void){return tos==0;}           //Is the Stack empty?
    void Push(T v)                               //Put value v on top of the Stack
    {   if (IsFull()) throw StackException("full");
        Element* n=new Element(tos,v);
        tos=n;
    }
    T Pop(void)                                  //Remove and return the value on top
    of
    {   if (IsEmpty()) throw StackException("empty");   //the Stack
        Element* p=tos;
        T v=p->value;
        tos=p->prev;
        delete p;
        return v;
    }
};
#endif
```

Figure 2. File **Stack.h**

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
#include "Stack.h"
int main()
{    string t="tilt";
     Stack <char> S;
     for (int a=0;a<t.length();a++) S.Push(t[a]);
     cout << endl;
     while (!S.IsEmpty()) cout << S.Pop();
     cout << endl;
     return 0;
}
```

Figure 3. File **p01.cpp**

Program **p01** prints:
tlit

Exercises:

1. Modify file **p01.cpp** given in this note so that the stack stores values of type **double**. Write a program that will evaluate a postfix string. For example the postfix string "3 4 5 * +" evaluates to 60.
2. A queue is a first-in-first-out (FIFO) data structure. Design a dequeue (pronounced "deck") that implements both a stack and a queue as a template.