Feb 15 15:15 1997 driver.dem Page 1

```
% File "driver.dem".
% Used to test stack1.tu and stack2.tu
% Author: Tim Brecht, January, 1997
%         Note that there aren't any interactive ways
%         to call Initialize and Finalize and the specification didn't
%         require it (it would probably be a good idea to provide ways
%         to test these).

% Just to show that there are many different implementations.
% import Stack in "debug.tu"
import Stack in "stack2.tu"

type elementType : int

var cmdstr  : string
var str     : elementType

Stack.Initialize


loop
    put "Enter command: " ..
    % I allow an entire string to be entered and just use
    % the first character (this has its pros and cons)
    get cmdstr
    put " Command is: ", cmdstr(1)

    case cmdstr(1) of
        % used to quit
        label 'q' :
             put "Quiting"
             exit

        % push an element - prompt the user for the element to push
        label 'P' :
             % check that the stack isn't full before Pushing
             if Stack.IsFull then
                 put "Stack is full - can't push another element"
             else
                 put "Enter integer to push: " ..
                 get str
                 put "Pushing ", str
                 Stack.Push(str)
             end if

        % pop the top element
        label 'p' :
             % check that the stack isn't empty before Pushing
             if Stack.IsEmpty then
                 put "Stack is empty - can't pop an element"
             else
                 Stack.Pop(str)
                 put "Popped: ", str
             end if

        % show / print the contents of the stack
```


Feb 15 15:15 1997 driver.dem Page 2

```
        label 's' :
             % Not required for the first part.
             % Stack.Show
             put "Not implemented"

        % check if the stack is empty
        label 'e' :
             if (Stack.IsEmpty) then
                 put "Stack is Empty"
             else
                 put "Stack is not Empty"
             end if

        % check if the stack is full
        label 'f' :
             if (Stack.IsFull) then
                 put "Stack is Full"
             else
                 put "Stack is not Full"
             end if

        % inspect top - print the top element of the stack
        label 't' :
             % ensure that there is something on the stack to inspect
             if Stack.IsEmpty then
                 put "Stack is empty - can't inspect top element"
             else
                 put "Top Element is: ", Stack.InspectTop
             end if

        % help - print a list of available commands
        label '?', 'h' :
             put "  P - Push"
             put "  p - pop"
             put "  q - quit"
             put "  s - show all of stack (print)"
             put "  e - is empty"
             put "  f - is full"
             put "  t - inspect top"
             put "  ? - print list of commands"
             put "  h - print list of commands"

        % user enters a command that we don't know about
        label :
             put "Command not found"
             put "Type ? for help"

    end case

end loop

% clean up / free up any space used by the stack
Stack.Finalize
put "Driver Program Finished"
```


Jan 15 11:16 1997 tester.dem Page 1

```
% File "tester.dem".
% Used to test stack1.tu and stack2.tu
% Author: Tim Brecht, January, 1997
%   This is a basic implementation.
%   It does a reasonable job but not an adequate job.
%   See "tester2.dem"

import Stack in "../stack1.tu"
% import Stack in "../stack2.tu"

var value : int

Stack.Initialize

assert(Stack.IsEmpty)
Stack.Push (1)
assert(Stack.InspectTop = 1)
Stack.Pop (value)
assert(value = 1)
assert(Stack.IsEmpty)
Stack.Push(1)
assert(Stack.InspectTop = 1)
assert(Stack.IsEmpty = false)
Stack.Push(2)
assert(Stack.InspectTop = 2)
Stack.Push(3)
assert(Stack.InspectTop = 3)
Stack.Push(4)
assert(Stack.InspectTop = 4)
Stack.Push(5)
assert(Stack.InspectTop = 5)
assert(Stack.IsEmpty = false)
% Warning - only true for stack1.tu
assert(Stack.IsFull)

assert(Stack.InspectTop = 5)
Stack.Pop (value)
assert(value = 5)
assert(Stack.IsFull = false)
assert(Stack.IsEmpty = false)

assert(Stack.InspectTop = 4)
Stack.Pop (value)
assert(value = 4)
assert(Stack.IsFull = false)
assert(Stack.IsEmpty = false)

assert(Stack.InspectTop = 3)
Stack.Pop (value)
assert(value = 3)
assert(Stack.IsFull = false)
assert(Stack.IsEmpty = false)

assert(Stack.InspectTop = 2)
Stack.Pop (value)
assert(value = 2)
```


Jan 15 11:16 1997 tester.dem Page 2

```
assert(Stack.IsFull = false)
assert(Stack.IsEmpty = false)

assert(Stack.InspectTop = 1)
Stack.Pop (value)
assert(value = 1)
assert(Stack.IsFull = false)
assert(Stack.IsEmpty)

Stack.Finalize

assert(Stack.IsEmpty)
put "Test Succeeded"
```

```
% File "tester2.dem".
% Used to test stack1.tu and stack2.tu
% Author: Tim Brecht, January, 1997
%   This is a muct better implementation than tester.dem.
%   It not only does a good job of testing but also
%   works around bugs found in one of the implementations
%   in order to push the testing to the limits.
%   (Working around the bugs is not required but it does
%   allow full testing of all of the other features).
%   Another nice feature is the ability to turn on tracing
%   to help track the testing (so you know what stage is
%   being tested).

import Stack in "../stack1.tu"
% import Stack in "../stack2.tu"

const AVOID_BUGS  := true   % stack1 has bugs in finalize - this avoids them
                            % *** set this to false when testing stack2 ***
const TRACE       := false  % gives feedback related to progress on tests
                            % should be false for final testing
const MaxElements := 50     % maximum number of elements to put into stack

var   numelements : int

% put a bunch of things on the stack
% either filling it or stopping at a reasonable
% and easy to change limit

function FillStack(NumElements : int) : int
    var popped : int
    var value  : int

    value := 1
    loop
        if Stack.IsFull or value > NumElements then
            % Provide some feedback (if wanted/needed)
            if TRACE then
                if Stack.IsFull then
                    put "Stack was filled with ", value-1, " elements"
                else
                    put "Stack filling stopped at ", value-1, " elements"
                end if
            end if
            exit
        else
            % check IsFull
            assert(Stack.IsFull = false)

            % push the value and ensure it's at the top
            Stack.Push(value)
            assert(Stack.InspectTop = value)

            % check that pop and push work as advertised
            Stack.Pop(popped)
            assert(popped = value)
            Stack.Push(popped)
```



```
            assert(Stack.InspectTop = value)

            % check IsEmpty
            assert(Stack.IsEmpty = false)
        end if
        value := value + 1
    end loop
    assert(Stack.InspectTop = value-1)
    assert(Stack.InspectTop = NumElements or Stack.IsFull)
    result value-1
end FillStack

if TRACE then
    put "About to Initialize the Stack"
end if
% stack should be empty and should not be full :-)
Stack.Initialize
assert(Stack.IsEmpty)
assert(Stack.IsFull = false)

if TRACE then
    put "About to Fill the Stack"
end if
% should be full or at the limit
numelements := FillStack(MaxElements)
assert(numelements = MaxElements or Stack.IsFull)
assert(Stack.IsEmpty = false)

if TRACE then
    put "About to Initialize the Stack"
end if
% check that initialize works correctly
% stack should be empty and should not be full :-)
Stack.Initialize
assert(Stack.IsEmpty)
assert(Stack.IsFull = false)

if TRACE then
    put "About to Fill the Stack"
end if
% fill up the stack again
% should be full or at the limit
numelements := FillStack(MaxElements)
assert(numelements = MaxElements or Stack.IsFull)
assert(Stack.IsEmpty = false)

if TRACE then
    put "About to Finalize the Stack"
end if
% check that Finalize works correctly
Stack.Finalize
if AVOID_BUGS then
    Stack.Initialize
    if TRACE then
        put " ... Called Initialize due to bugs in Finalize"
    end if
```



```
end if

% NOTE: This these fail for stack1.tu without calling
% initialize after finalize (that bug also causes subsequent failures)
assert(Stack.IsEmpty)
assert(Stack.IsFull = false)

if TRACE then
    put "About to Fill the Stack"
end if
% check that we can add correctly after finalize
% should be full or at the limit
numelements := FillStack(MaxElements)
assert(numelements = MaxElements or Stack.IsFull)
assert(Stack.IsEmpty = false)

if TRACE then
    put "About to Finalize the Stack and end"
end if
% Now clean up everything before ending
% check that Finalize works correctly
Stack.Finalize

% ** Now we go ahead and let the assertions fail since we've
% ** tested everything else.
% ** This will fail with stack1.tu but not with stack2.tu
%    It would be a mistake to totally avoid these bugs
%    since they must be reported.
assert(Stack.IsEmpty)
assert(Stack.IsFull = false)

put "Test Succeeded"
```



```
unit
module Stack
export Push, Pop, InspectTop, IsEmpty, IsFull, Initialize, Finalize

type elementType : int
const NOTHING    := -9999

    % Add element to the top of the stack. The stack must not be full.
    procedure Push (element : elementType)
        put "Pushing ", element
    end Push

    % Remove element from top of the stack. The stack must not be empty.
    procedure Pop (var element : elementType)
        put "Popping "
        element := NOTHING
    end Pop

    % Return element from top of stack, without changing the stack.
    % The stack must not be empty.
    function InspectTop : elementType
         put "Inspecting Top"
         result NOTHING
    end InspectTop

    % Return true if the stack contains no elements.
    function IsEmpty : boolean
         put "IsEmpty "
         result true
    end IsEmpty

    % Return true if the stack cannot hold any more elements.
    function IsFull : boolean
         put "IsFull "
         result false
    end IsFull

    % Initialize the stack so it can be used. Make it be empty.
    procedure Initialize
        put "Initializing "
    end Initialize

    % Clear the stack, freeing any space allocated for its elements.
    procedure Finalize
        put "Finalizing "
    end Finalize

end Stack
```