

Bug Free Coding with SPARK
Ada
Release 2024-09

Robert Tice

Sep 08, 2024

CONTENTS:

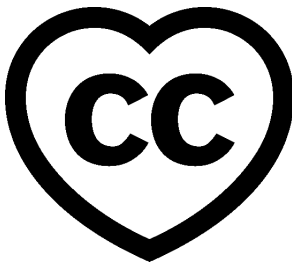
| | | |
|----------|----------------------------|----------|
| 1 | Let's Build a Stack | 3 |
| 1.1 | Background | 3 |
| 1.2 | Input Format | 6 |
| 1.3 | Constraints | 6 |
| 1.4 | Output Format | 6 |
| 1.5 | Sample Input | 6 |
| 1.6 | Sample Output | 6 |

Warning

This version of the website contains UNPUBLISHED contents. Please do not share it externally!

Copyright © 2018 - 2022, AdaCore

This book is published under a CC BY-SA license, which means that you can copy, redistribute, remix, transform, and build upon the content for any purpose, even commercially, as long as you give appropriate credit, provide a link to the license, and indicate if changes were made. If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. You can find license details [on this page](#)¹



Workshop project: Learn to write maintainable bug-free code with SPARK Ada.

This document was written by Robert Tice.

Note

The code examples in this course use an 80-column limit, which is a typical limit for Ada code. Note that, on devices with a small screen size, some code examples might be difficult to read.

¹ <http://creativecommons.org/licenses/by-sa/4.0>

LET'S BUILD A STACK

In this lab we will build a stack data structure and use the SPARK provers to find the errors in the below implementation.

1.1 Background

So, what is a stack?

A stack is like a pile of dishes...



1. The pile starts out empty.
2. You add (push) a new plate (data) to the stack by placing it on the top of the pile.
3. To get plates (data) out, you take the one off the top of the pile (pop).
4. Our stack has a maximum height (size) of 9 dishes

Pushing items onto the stack

Here's what should happen if we pushed the string MLH onto the stack.

Step 0:

Empty

| | |
|----|--|
| 1: | |
| 2: | |
| 3: | |
| 4: | |
| 5: | |

Last = 0

Step 1:

Push("M")

| | |
|----|---|
| 1: | M |
| 2: | |
| 3: | |
| 4: | |
| 5: | |

Last = 1

Step 2:

Push("L")

| | |
|----|---|
| 1: | M |
| 2: | L |
| 3: | |
| 4: | |
| 5: | |

Last = 2

Step 3:

Push("H")

| | |
|----|---|
| 1: | M |
| 2: | L |
| 3: | H |
| 4: | |
| 5: | |

Last = 3

Step 4:

Top()

| | |
|----|---|
| 1: | M |
| 2: | L |
| 3: | H |
| 4: | |
| 5: | |

Last = 3

returns:

'H'

The list starts out empty. Each time we push a character onto the stack, Last increments by 1.

Popping items from the stack

Here's what should happen if we popped 2 characters off our stack & then clear it.

Step 0:

Start

| | |
|----|---|
| 1: | M |
| 2: | L |
| 3: | H |
| 4: | |
| 5: | |

Last = 3

Step 1:

Pop()

| | |
|----|---|
| 1: | M |
| 2: | L |
| 3: | H |
| 4: | |
| 5: | |

Last = 2

returns:

'H'

Step 2:

Pop()

| | |
|----|---|
| 1: | M |
| 2: | L |
| 3: | H |
| 4: | |
| 5: | |

Last = 1

returns:

'L'

Step 3:

Clear()

| | |
|----|---|
| 1: | M |
| 2: | L |
| 3: | H |
| 4: | |
| 5: | |

Last = 0

Note that pop and clear don't unset the Storage array's elements, they just change the value of Last.

1.2 Input Format

N inputs will be read from stdin/console as inputs, C to the stack.

1.3 Constraints

$1 \leq N \leq 1000$

C is any character. Characters d and p will be special characters corresponding to the below commands:

p => Pops a character off the stack

d => Prints the current characters in the stack

1.4 Output Format

If the stack currently has the characters "M", "L", and "H" then the program should print the stack like this:

[M, L, H]

1.5 Sample Input

M L H d p d p d p d

1.6 Sample Output

[M, L, H] [M, L] [M] []

Listing 1: stack.ads

```
1 package Stack with SPARK_Mode => On is
2
3   procedure Push (V : Character)
4     with Pre => not Full,
5           Post => Size = Size'Old + 1;
6
7   procedure Pop (V : out Character)
8     with Pre => not Empty,
9           Post => Size = Size'Old - 1;
10
11  procedure Clear
12    with Post => Size = 0;
13
14  function Top return Character
15    with Post => Top'Result = Tab>Last);
16
17  Max_Size : constant := 9;
```

(continues on next page)

(continued from previous page)

```

18  -- The stack size.
19
20  Last : Integer range 0 .. Max_Size := 0;
21  -- Indicates the top of the stack. When 0 the stack is empty.
22
23  Tab : array (1 .. Max_Size) of Character;
24  -- The stack. We push and pop pointers to Values.
25
26  function Full return Boolean is (Last = Max_Size);
27
28  function Empty return Boolean is (Last < 1);
29
30  function Size return Integer is (Last);
31
32 end Stack;

```

Listing 2: stack.adb

```

1  package body Stack with SPARK_Mode => On is
2
3      -----
4      -- Clear --
5      -----
6
7      procedure Clear
8      is
9      begin
10         Last := Tab'First;
11     end Clear;
12
13     -----
14     -- Push --
15     -----
16
17     procedure Push (V : Character)
18     is
19     begin
20         Tab (Last) := V;
21     end Push;
22
23     -----
24     -- Pop --
25     -----
26
27     procedure Pop (V : out Character)
28     is
29     begin
30         Last := Last - 1;
31         V := Tab (Last);
32     end Pop;
33
34     -----
35     -- Top --
36     -----
37
38     function Top return Character
39     is
40     begin
41         return Tab (1);
42     end Top;
43

```

(continues on next page)

```
44 end Stack;
```

Listing 3: main.adb

```

1 with Ada.Command_Line; use Ada.Command_Line;
2 with Ada.Text_IO;      use Ada.Text_IO;
3 with Stack;           use Stack;
4
5 procedure Main with SPARK_Mode => Off
6 is
7
8     -----
9     -- Debug --
10    -----
11
12    procedure Debug
13    is
14    begin
15
16        if not Stack.Empty then
17
18            Put ("[";
19            for I in Stack.Tab'First .. Stack.Size - 1 loop
20                Put (Stack.Tab (I) & ", ");
21            end loop;
22            Put_Line (Stack.Tab (Stack.Size) & "]");
23        else
24            Put_Line ("[]");
25        end if;
26
27    end Debug;
28
29    S : Character;
30
31 begin
32
33     -----
34     -- Main --
35     -----
36
37    for Arg in 1 .. Argument_Count loop
38        if Argument (Arg)'Length /= 1 then
39            Put_Line (Argument (Arg) & " is an invalid input to the stack.");
40        else
41            S := Argument (Arg)(Argument (Arg)'First);
42
43            if S = 'd' then
44                Debug;
45            elsif S = 'p' then
46                if not Stack.Empty then
47                    Stack.Pop (S);
48                else
49                    Put_Line ("Nothing to Pop, Stack is empty!");
50                end if;
51            else
52                if not Stack.Full then
53                    Stack.Push (S);
54                else
55                    Put_Line ("Could not push '" & S & "', Stack is full!");
56                end if;
57            end if;

```

(continues on next page)

(continued from previous page)

```
58     end if;  
59  
60     end loop;  
61  
62 end Main;
```