Rutger A. Yager

Dr. Burris COSC 3319.01

October 4th, 2013

LAB 1, GRADE OPTION “A”

**Input and Output results:**

**Lab1\_Input.txt**

C

20

Joe;

moT;

Betty;

pop;

pop;

pop;

pop;

Larry;

Sarah;

Bob;

pop;

Harold;

Sue;

exit;

I

15

10 25 3 46 0;

pop;

32 12 456 7 9 0;

5 17 0;

3 0;

42 23 67 0;

pop;

pop;

32 22 56 78 54 6 17 0;

exit;

D

8

January 15 1952 February 23 1492 September 1 2004 end;

October 12 2003 end;

February 4 1777 December 24 2003 June 12 2004 end;

pop;

pop;

pop;

pop;

September 17 1623 August 11 2002 end;

pop;

exit;

done

**Lab1\_Output.txt**

The following file was generated with the command  
**RadioTelescope < Lab1\_Input.txt > Lab1\_Output.txt**

Enter a Stack Type (C/I/D): Enter a Stack Size: Successfully allocated a character stack of size 20.

Type a stream of characters followed by spaces to insert.

Type 'pop;' to pop a stream of characters. Type 'exit;' to exit.

Inserted -Joe- into the stack. Current size is now 4.

Inserted -moT- into the stack. Current size is now 8.

Inserted -Betty- into the stack. Current size is now 14.

Pop results: 5ytteB

Pop results: 3Tom

Pop results: 3eoJ

Nothing to pop!

Inserted -Larry- into the stack. Current size is now 6.

Inserted -Sarah- into the stack. Current size is now 12.

Inserted -Bob- into the stack. Current size is now 16.

Pop results: 3boB

Inserted -Harold- into the stack. Current size is now 19.

No space left in Stack, could not insert!

Enter a Stack Type (C/I/D): Enter a Stack Size: Successfully allocated an integer stack of size 15.

Enter a stream of integers separated by spaces.

The last integer must be '0' followed by ';'.

Type 'pop;' to pop a stream of integers. Type 'exit;' to exit.

Inserted 4 integers into the stack. Size is now 5.

Pop results: 4 46 3 25 10

Inserted 5 integers into the stack. Size is now 6.

Inserted 2 integers into the stack. Size is now 9.

Inserted 1 integers into the stack. Size is now 11.

Inserted 3 integers into the stack. Size is now 15.

Pop results: 3 67 23 42

Pop results: 1 3

No space left in stack, could not insert!

Enter a Stack Type (C/I/D): Enter a Stack Size: Successfully allocated a date stack of size 8.

Enter a stream of dates separated by spaces in the following format:

Month Day Year

The last word in the stream must be 'end' followed by ';'.

Type 'pop;' to pop a stream of dates. Type 'exit;' to exit;

Inserted 3 Dates into the stack. Size is now 4.

Inserted 1 Dates into the stack. Size is now 6.

No space left in stack, could not insert!

1 Dates:

OCTOBER 12, 2003

3 Dates:

SEPTEMBER 1, 2004

FEBRUARY 23, 1492

JANUARY 15, 1952

Nothing to pop!

Nothing to pop!

Inserted 2 Dates into the stack. Size is now 3.

2 Dates:

AUGUST 11, 2002

SEPTEMBER 17, 1623

Enter a Stack Type (C/I/D):

**Source Files**

**RadioTelescope.adb**

with Ada.Text\_IO;

use Ada.Text\_IO;

with Ada.Strings.Unbounded.Text\_IO;

use Ada.Strings.Unbounded.Text\_IO;

with Ada.Characters.Latin\_1;

use Ada.Characters.Latin\_1;

with Ada.Integer\_Text\_IO;

use Ada.Integer\_Text\_IO;

with Ada.Strings.Unbounded;

use Ada.Strings.Unbounded;

with Unchecked\_Conversion;

with GStack;

procedure RadioTelescope is

type StackType is

(C,

I,

D,

Done);

type MonthName is

(January,

February,

March,

April,

May,

June,

July,

August,

September,

October,

November,

December);

type Date is

record

Month : MonthName;

Day : Integer range 1 .. 31;

Year : Integer;

end record;

package Month\_IO is new Ada.Text\_IO.Enumeration\_IO(MonthName);

use Month\_IO;

package Type\_IO is new Ada.Text\_IO.Enumeration\_IO(StackType);

use Type\_IO;

function Character\_To\_Int is

new Unchecked\_Conversion(Character, Integer);

function Int\_To\_Character is

new Unchecked\_Conversion(Integer, Character);

function Int\_To\_Date is

new Unchecked\_Conversion(Integer, Date);

function Date\_To\_Int is

new Unchecked\_Conversion(Date, Integer);

Size : Natural;

Stack : StackType;

begin

Main:

loop

Put("Enter a Stack Type (C/I/D): ");

Get(Stack);

if Stack /= Done then

Put("Enter a Stack Size: ");

Get(Size);

end if;

case Stack is

when C =>

declare

package Character\_Stack is new GStack(CHARACTER, Size);

use Character\_Stack;

procedure GetInputString (

A\_String : out Unbounded\_String) is

Char : Character;

begin

loop

Get(Char);

exit when Char = Semicolon or Char = Space;

Append(A\_String, Char);

end loop;

end GetInputString;

procedure InsertCharacters (

A\_String : in String) is

Succeed : Boolean;

begin

if (A\_String'Length + Current\_Size) >= Total\_Size then

Put("No space left in Stack, could not insert!");

New\_Line;

else

for I in 1..A\_String'Length loop

Push(A\_String(I), Succeed);

if Succeed /= True then

Put("Failed to insert '");

Put(A\_String(I));

Put\_Line("'.");

end if;

end loop;

Push(Int\_To\_Character(A\_String'Length), Succeed);

if Succeed /= True then

Put\_Line("Could not load stream length.");

end if;

Put("Inserted -");

Put(A\_String);

Put("- into the stack. Current size is now ");

Put(Current\_Size, 2);

Put(".");

New\_Line;

end if;

end InsertCharacters;

procedure PopCharacters is

Count : Integer;

Temp : Character;

Succeed : Boolean;

begin

if Current\_Size = 0 then

Put\_Line("Nothing to pop!");

else

Put("Pop results: ");

Pop(Temp, Succeed);

if Succeed then

Put(Character\_To\_Int(Temp), 2);

Count := Character\_To\_Int(Temp);

for I in 1..Count loop

Pop(Temp, Succeed);

if Succeed then

Put(Temp);

else

New\_Line;

Put("Could not retrieve character ");

Put(I, 2);

New\_Line;

end if;

end loop;

New\_Line;

else

Put\_Line("Failed to retrieve character stream.");

end if;

end if;

end Popcharacters;

begin

Put("Successfully allocated a character stack of size ");

Put(Size, 2);

Put\_Line(".");

Put\_Line(

"Type a stream of characters followed by spaces to insert.");

Put\_Line(

"Type 'pop;' to pop a stream of characters. Type 'exit;' to exit.");

loop

declare

Temp : Unbounded\_String;

begin

GetInputString(Temp);

declare

Input : String (1 .. Length (Temp));

begin

Input := To\_String(Temp);

if Input = "pop" then

PopCharacters;

elsif Input = "exit" then

exit;

else

InsertCharacters(Input);

end if;

end;

end;

end loop;

end;

when I =>

declare

package Integer\_Stack is new GStack(INTEGER, Size);

use Integer\_Stack;

type Int\_Array is array (Positive range <>) of Integer;

procedure GetInputString (

A\_String : out Unbounded\_String) is

Char : Character;

begin

loop

Get(Char);

exit when Char = Semicolon or Char = Space;

Append(A\_String, Char);

end loop;

end GetInputString;

procedure RecordInteger (

IntArray : in out Int\_Array;

Index : in out Integer;

An\_Int : in Integer) is

begin

Index := Index + 1;

IntArray(Index) := An\_Int;

end RecordInteger;

procedure PushInts (

IntArray : in Int\_Array;

Count : in Integer) is

Succeed : Boolean;

begin

if Count + Current\_Size >= Total\_Size then

Put\_Line("No space left in stack, could not insert!");

return;

end if;

for I in 1..Count loop

Push(IntArray(I), Succeed);

if Succeed /= True then

Put("Failed to insert ");

Put(IntArray(I), 3);

Put\_Line(".");

end if;

end loop;

Push(Count, Succeed);

if Succeed /= True then

Put\_Line("Could not load stream length.");

end if;

Put("Inserted ");

Put(Count, 2);

Put(" integers into the stack. Size is now ");

Put(Current\_Size, 2);

Put\_Line(".");

end PushInts;

procedure PopIntegers is

Count : Integer;

Int : Integer;

Succeed : Boolean;

begin

if Current\_Size = 0 then

Put\_Line("Nothing to pop!");

return;

end if;

Pop(Count, Succeed);

if Succeed then

Put("Pop results: ");

Put(Count, 4);

for I in 1..Count loop

Pop(Int, Succeed);

if Succeed /= True then

Put("Could not retrieve Integer #");

Put(I, 2);

Put\_Line(".");

else

Put(Int, 4);

end if;

end loop;

New\_Line;

else

Put\_Line("Could not retrieve Integers.");

end if;

end PopIntegers;

Value : Integer;

Count : Integer := 0;

Ints : Int\_Array (1 .. Size);

begin

Put("Successfully allocated an integer stack of size ");

Put(Size, 2);

Put\_Line(".");

Put\_Line("Enter a stream of integers separated by spaces.");

Put("The last integer must be '0' followed by ';'.");

New\_Line;

Put(

"Type 'pop;' to pop a stream of integers. Type 'exit;' to exit.");

New\_Line;

loop

declare

Temp : Unbounded\_String;

begin

GetInputString(Temp);

declare

Input : String (1 .. Length (Temp));

begin

Input := To\_String(Temp);

if Input = "pop" then

PopIntegers;

elsif Input = "exit" then

exit;

else

Value := Integer'Value(Input);

if Value = 0 then

PushInts(Ints, Count);

Count := 0;

else

RecordInteger(Ints, Count, Value);

end if;

end if;

end;

end;

end loop;

end;

when D =>

declare

package Date\_Stack is new GStack(Date, Size);

use Date\_Stack;

type Date\_Array is array (Positive range <>) of Date;

procedure PrintDate (

A\_Date : in Date) is

begin

Put(A\_Date.Month);

Put(A\_Date.Day, 3);

Put(", ");

Put(A\_Date.Year, 4);

end PrintDate;

procedure GetInputString (

A\_String : out Unbounded\_String) is

Char : Character;

begin

loop

Get(Char);

exit when Char = Semicolon or Char = Space;

Append(A\_String, Char);

end loop;

end GetInputString;

procedure RecordDate (

Some\_Dates : in out Date\_Array;

Index : in out Integer;

A\_Date : in Date) is

begin

Index := Index + 1;

Some\_Dates(Index) := A\_Date;

end;

procedure PushDates (

Some\_Dates : in Date\_Array;

Count : in Integer) is

Succeed : Boolean;

begin

if Count + Current\_Size >= Total\_Size then

Put\_Line("No space left in stack, could not insert!");

return;

end if;

for I in 1..Count loop

Push(Some\_Dates(I), Succeed);

if Succeed /= True then

Put\_Line("Failed to insert date.");

end if;

end loop;

Push(Int\_To\_Date(Count), Succeed);

if Succeed /= True then

Put\_Line("Could not load stream length.");

end if;

Put("Inserted ");

Put(Count, 2);

Put(" Dates into the stack. Size is now ");

Put(Current\_Size, 2);

Put\_Line(".");

end;

procedure PopDates is

Count : Integer;

Temp : Date;

Succeed : Boolean;

begin

if Current\_Size = 0 then

Put\_Line("Nothing to pop!");

else

Pop(Temp, Succeed);

if Succeed then

Count := Date\_To\_Int(Temp);

Put(Count, 2);

Put\_Line(" Dates:");

for I in 1..Count loop

Pop(Temp, Succeed);

if Succeed then

PrintDate(Temp);

New\_Line;

else

Put("Could not retrieve date #");

Put(I, 2);

Put\_Line(".");

end if;

end loop;

else

Put\_Line("Could not retrieve dates.");

end if;

end if;

end PopDates;

Control : Integer := 0;

Count : Integer := 0;

Dates : Date\_Array (1 .. Size);

Temp\_Date : Date;

begin

Put("Successfully allocated a date stack of size ");

Put(Size, 2);

Put\_Line(".");

Put\_Line(

"Enter a stream of dates separated by spaces in the following format:");

Put\_Line("Month Day Year");

Put\_Line(

"The last word in the stream must be 'end' followed by ';'.");

Put\_Line(

"Type 'pop;' to pop a stream of dates. Type 'exit;' to exit;");

loop

declare

Temp : Unbounded\_String;

begin

GetInputString(Temp);

declare

Input : String (1 .. Length (Temp));

begin

Input := To\_String(Temp);

if Input = "pop" then

PopDates;

elsif Input = "end" then

PushDates(Dates, Count);

Count := 0;

elsif Input = "exit" then

exit;

else

case Control is

when 0 =>

Temp\_Date.Month := MonthName'Value(Input);

Control := 1;

when 1=>

Temp\_Date.Day := Integer'Value(Input);

Control := 2;

when 2 =>

Temp\_Date.Year := Integer'Value(Input);

RecordDate(Dates, Count, Temp\_Date);

Control := 0;

when others =>

null;

end case;

end if;

end;

end;

end loop;

end;

when Done =>

exit;

end case;

end loop Main;

end RadioTelescope;

**GStack.ads**

generic

type ITEM is private;

Size : Positive := 5;

package GStack is

procedure Push (

Stuff : in ITEM;

Succeed : out BOOLEAN);

procedure Pop (

Stuff : out ITEM;

Succeed : out BOOLEAN);

function Is\_Empty return BOOLEAN;

function Is\_Full return BOOLEAN;

function Current\_Size return NATURAL;

function Total\_Size return POSITIVE;

end GStack;

**GStack.adb**

package body GStack is

StackArray : array (1 .. Size) of ITEM;

Top : Natural := 0;

function Is\_Empty return BOOLEAN is

begin

return Top = 0;

end Is\_Empty;

function Is\_Full return BOOLEAN is

begin

return Top = Size;

end Is\_Full;

procedure Push (

Stuff : in ITEM;

Succeed : out BOOLEAN) is

begin

if Is\_Full then

Succeed := False;

return;

end if;

Top := Top + 1;

StackArray(Top) := Stuff;

Succeed := True;

end Push;

procedure Pop (

Stuff : out ITEM;

Succeed : out BOOLEAN) is

begin

if Is\_Empty then

Succeed := False;

return;

end if;

Stuff := StackArray(Top);

Top := Top - 1;

Succeed := True;

end Pop;

function Current\_Size return NATURAL is

begin

return Top;

end Current\_Size;

function Total\_Size return POSITIVE is

begin

return Size;

end;

end GStack;