Rutger Yager

Dr. Burris COSC 3319.01

10/25/2013

GRADE OPTION “A”

**INPUT AND RESULTS**

**Lab2in.txt:**

Names

-5

42

2

17

4

I1 Lee;

I1 Rios;

D1;

I1 Carazo;

I1 Hodges;

I3 Patil;

I2 Garza;

I3 Morgan;

D4;

I4 McLeod;

I4 Yager;

D3;

D4;

I1 Wills;

I3 Assi;

I1 Yager;

I1 Hall;

I4 Hodges;

I1 Rios;

I1 Lee;

I1 Assi;

D1;

D1;

D1;

I2 Pokhrel;

I4 Sido;

I4 Keo;

I2 Sodemann;

I4 Davies;

I4 Hodges;

I4 McLeod;

I4 Furbee;

I4 Garza;

I1 Sparks;

I1 Vincik;

I1 Dedear;

I2 Pokhrel;

I1 Assi;

D2;

D2;

D2;

I2 Guo;

I2 Assi;

I1 Doo;

I1 Burris;

exit;

Names

-5

42

15

39

3

I1 Lee;

I1 Rios;

D1;

I1 Carazo;

I1 Hodges;

I3 Patil;

I2 Garza;

I3 Morgan;

D3;

I1 Wills;

I3 Assi;

I1 Yager;

I1 Hall;

I1 Rios;

I1 Lee;

I1 Assi;

D1;

D1;

D1;

I2 Pokhrel;

I2 Sodemann;

I1 Sparks;

I1 Vincik;

I1 Dedear;

I2 Pokhrel;

I1 Assi;

D2;

D2;

D2;

I2 Guo;

I2 Assi;

I1 Doo;

I1 Burris;

exit;

Dates

0

50

0

20

3

I2 January 15 1956;

I2 February 14 1947;

I3 October 16 1986;

I2 September 17 1842;

I1 December 24 1996;

D3;

D1;

I3 March 16 1992;

D3;

D1;

I2 January 15 1956;

I3 April 14 1492;

I3 November 7 1776;

I3 June 12 1994;

I2 July 4 1776;

I2 January 15 2012;

I3 December 6 1991;

I3 March 5 1886;

I1 October 24 1996;

I1 November 23 1996;

I1 November 2 1990;

I3 September 14 1998;

exit;

done

**Lab2out.txt:**This file was generated with the command: stackstest < Lab2in.txt > Lab2out.txt

Enter a stack type (Names/Dates):

Enter the start of memory:

Enter the end of memory:

Enter the front of the stack list:

Enter the rear of the stack list:

Enter number of stacks:

Inserted -LEE- into stack 1.

Inserted -RIOS- into stack 1.

Pop Results for stack 1: RIOS

Inserted -CARAZO- into stack 1.

Inserted -HODGES- into stack 1.

Inserted -PATIL- into stack 3.

Inserted -GARZA- into stack 2.

Inserted -MORGAN- into stack 3.

Nothing to pop!

Inserted -MCLEOD- into stack 4.

Inserted -YAGER- into stack 4.

Pop Results for stack 3: MORGAN

Pop Results for stack 4: YAGER

OVERFLOW IN STACK 1!

Base [ 2, 5, 9, 13, 17]

Top [ 5, 6, 10, 14]

OldTop [ 2, 5, 9, 13]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES]

Stack 2 [ 6: GARZA]

Stack 3 [10: PATIL]

Stack 4 [14: MCLEOD]

REALLOCATING...

Base [ 2, 10, 12, 14, 17]

Top [ 6, 11, 13, 15]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS]

Stack 2 [11: GARZA]

Stack 3 [13: PATIL]

Stack 4 [15: MCLEOD]

Inserted -WILLS- into stack 1.

Inserted -ASSI- into stack 3.

Inserted -YAGER- into stack 1.

Inserted -HALL- into stack 1.

Inserted -HODGES- into stack 4.

Inserted -RIOS- into stack 1.

Inserted -LEE- into stack 1.

OVERFLOW IN STACK 1!

Base [ 2, 10, 12, 14, 17]

Top [10, 11, 14, 16]

OldTop [ 6, 11, 13, 15]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL, 9: RIOS, 10: LEE]

Stack 2 [11: GARZA]

Stack 3 [13: PATIL, 14: ASSI]

Stack 4 [15: MCLEOD, 16: HODGES]

REALLOCATING...

Base [ 2, 11, 12, 14, 17]

Top [11, 12, 14, 16]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL, 9: RIOS, 10: LEE, 11: ASSI]

Stack 2 [12: GARZA]

Stack 3 [13: PATIL, 14: ASSI]

Stack 4 [15: MCLEOD, 16: HODGES]

Inserted -ASSI- into stack 1.

Pop Results for stack 1: ASSI

Pop Results for stack 1: LEE

Pop Results for stack 1: RIOS

OVERFLOW IN STACK 2!

Base [ 2, 11, 12, 14, 17]

Top [ 8, 12, 14, 16]

OldTop [11, 12, 14, 16]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [12: GARZA]

Stack 3 [13: PATIL, 14: ASSI]

Stack 4 [15: MCLEOD, 16: HODGES]

REALLOCATING...

Base [ 2, 8, 12, 14, 17]

Top [ 8, 10, 14, 16]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [13: PATIL, 14: ASSI]

Stack 4 [15: MCLEOD, 16: HODGES]

Inserted -POKHREL- into stack 2.

Inserted -SIDO- into stack 4.

OVERFLOW IN STACK 4!

Base [ 2, 8, 12, 14, 17]

Top [ 8, 10, 14, 17]

OldTop [ 8, 10, 14, 16]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [13: PATIL, 14: ASSI]

Stack 4 [15: MCLEOD, 16: HODGES, 17: SIDO]

REALLOCATING...

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 16]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO]

Inserted -KEO- into stack 4.

OVERFLOW IN STACK 2!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 16]

OldTop [ 8, 10, 12, 16]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert SODEMANN into stack 2<<

Inserted -DAVIES- into stack 4.

OVERFLOW IN STACK 4!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 16]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert HODGES into stack 4<<

OVERFLOW IN STACK 4!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert MCLEOD into stack 4<<

OVERFLOW IN STACK 4!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert FURBEE into stack 4<<

OVERFLOW IN STACK 4!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert GARZA into stack 4<<

OVERFLOW IN STACK 1!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert SPARKS into stack 1<<

OVERFLOW IN STACK 1!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert VINCIK into stack 1<<

OVERFLOW IN STACK 1!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert DEDEAR into stack 1<<

OVERFLOW IN STACK 2!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert POKHREL into stack 2<<

OVERFLOW IN STACK 1!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GARZA, 10: POKHREL]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert ASSI into stack 1<<

Pop Results for stack 2: POKHREL

Pop Results for stack 2: GARZA

Nothing to pop!

Inserted -GUO- into stack 2.

Inserted -ASSI- into stack 2.

OVERFLOW IN STACK 1!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GUO, 10: ASSI]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert DOO into stack 1<<

OVERFLOW IN STACK 1!

Base [ 2, 8, 10, 12, 17]

Top [ 8, 10, 12, 17]

OldTop [ 8, 10, 12, 17]

Stack 1 [ 3: LEE, 4: CARAZO, 5: HODGES, 6: WILLS, 7: YAGER, 8: HALL]

Stack 2 [ 9: GUO, 10: ASSI]

Stack 3 [11: PATIL, 12: ASSI]

Stack 4 [13: MCLEOD, 14: HODGES, 15: SIDO, 16: KEO, 17: DAVIES]

REALLOCATING...

Could Not Reallocate: Not Enough Space.

>>Failed to insert BURRIS into stack 1<<

Enter a stack type (Names/Dates):

Enter the start of memory:

Enter the end of memory:

Enter the front of the stack list:

Enter the rear of the stack list:

Enter number of stacks:

Inserted -LEE- into stack 1.

Inserted -RIOS- into stack 1.

Pop Results for stack 1: RIOS

Inserted -CARAZO- into stack 1.

Inserted -HODGES- into stack 1.

Inserted -PATIL- into stack 3.

Inserted -GARZA- into stack 2.

Inserted -MORGAN- into stack 3.

Pop Results for stack 3: MORGAN

Inserted -WILLS- into stack 1.

Inserted -ASSI- into stack 3.

Inserted -YAGER- into stack 1.

Inserted -HALL- into stack 1.

Inserted -RIOS- into stack 1.

Inserted -LEE- into stack 1.

OVERFLOW IN STACK 1!

Base [15, 23, 31, 39]

Top [23, 24, 33]

OldTop [15, 23, 31]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL, 22: RIOS, 23: LEE]

Stack 2 [24: GARZA]

Stack 3 [32: PATIL, 33: ASSI]

REALLOCATING...

Base [15, 32, 34, 39]

Top [24, 33, 36]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL, 22: RIOS, 23: LEE, 24: ASSI]

Stack 2 [33: GARZA]

Stack 3 [35: PATIL, 36: ASSI]

Inserted -ASSI- into stack 1.

Pop Results for stack 1: ASSI

Pop Results for stack 1: LEE

Pop Results for stack 1: RIOS

Inserted -POKHREL- into stack 2.

OVERFLOW IN STACK 2!

Base [15, 32, 34, 39]

Top [21, 34, 36]

OldTop [24, 33, 36]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL]

Stack 2 [33: GARZA, 34: POKHREL]

Stack 3 [35: PATIL, 36: ASSI]

REALLOCATING...

Base [15, 21, 36, 39]

Top [21, 24, 38]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL]

Stack 2 [22: GARZA, 23: POKHREL, 24: SODEMANN]

Stack 3 [37: PATIL, 38: ASSI]

Inserted -SODEMANN- into stack 2.

OVERFLOW IN STACK 1!

Base [15, 21, 36, 39]

Top [21, 24, 38]

OldTop [21, 24, 38]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL]

Stack 2 [22: GARZA, 23: POKHREL, 24: SODEMANN]

Stack 3 [37: PATIL, 38: ASSI]

REALLOCATING...

Base [15, 33, 36, 39]

Top [22, 36, 38]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL, 22: SPARKS]

Stack 2 [34: GARZA, 35: POKHREL, 36: SODEMANN]

Stack 3 [37: PATIL, 38: ASSI]

Inserted -SPARKS- into stack 1.

Inserted -VINCIK- into stack 1.

Inserted -DEDEAR- into stack 1.

OVERFLOW IN STACK 2!

Base [15, 33, 36, 39]

Top [24, 36, 38]

OldTop [22, 36, 38]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL, 22: SPARKS, 23: VINCIK, 24: DEDEAR]

Stack 2 [34: GARZA, 35: POKHREL, 36: SODEMANN]

Stack 3 [37: PATIL, 38: ASSI]

REALLOCATING...

Base [15, 29, 36, 39]

Top [24, 33, 38]

Stack 1 [16: LEE, 17: CARAZO, 18: HODGES, 19: WILLS, 20: YAGER, 21: HALL, 22: SPARKS, 23: VINCIK, 24: DEDEAR]

Stack 2 [30: GARZA, 31: POKHREL, 32: SODEMANN, 33: POKHREL]

Stack 3 [37: PATIL, 38: ASSI]

Inserted -POKHREL- into stack 2.

Inserted -ASSI- into stack 1.

Pop Results for stack 2: POKHREL

Pop Results for stack 2: SODEMANN

Pop Results for stack 2: POKHREL

Inserted -GUO- into stack 2.

Inserted -ASSI- into stack 2.

Inserted -DOO- into stack 1.

Inserted -BURRIS- into stack 1.

Enter a stack type (Names/Dates):

Enter the start of memory:

Enter the end of memory:

Enter the front of the stack list:

Enter the rear of the stack list:

Enter number of stacks:

Inserted -JANUARY 15, 1956- into stack 2.

Inserted -FEBRUARY 14, 1947- into stack 2.

Inserted -OCTOBER 16, 1986- into stack 3.

Inserted -SEPTEMBER 17, 1842- into stack 2.

Inserted -DECEMBER 24, 1996- into stack 1.

Pop Results for stack 3: OCTOBER 16, 1986

Pop Results for stack 1: DECEMBER 24, 1996

Inserted -MARCH 16, 1992- into stack 3.

Pop Results for stack 3: MARCH 16, 1992

Nothing to pop!

Inserted -JANUARY 15, 1956- into stack 2.

Inserted -APRIL 14, 1492- into stack 3.

Inserted -NOVEMBER 7, 1776- into stack 3.

Inserted -JUNE 12, 1994- into stack 3.

Inserted -JULY 4, 1776- into stack 2.

Inserted -JANUARY 15, 2012- into stack 2.

Inserted -DECEMBER 6, 1991- into stack 3.

Inserted -MARCH 5, 1886- into stack 3.

Inserted -OCTOBER 24, 1996- into stack 1.

Inserted -NOVEMBER 23, 1996- into stack 1.

Inserted -NOVEMBER 2, 1990- into stack 1.

Inserted -SEPTEMBER 14, 1998- into stack 3.

Enter a stack type (Names/Dates):

**SOURCE CODE**

**Stackstest.adb:**

with Ada.Text\_IO;

use Ada.Text\_IO;

with Ada.Integer\_Text\_IO;

use Ada.Integer\_Text\_IO;

with Ada.Strings.Unbounded;

use Ada.Strings.Unbounded;

with Dynamicstacks;

with Reallocate;

with MoveStacks;

procedure StacksTest is

type Name is

(Lee,

Rios,

Carazo,

Hodges,

Patil,

Garza,

Morgan,

McLeod,

Yager,

Wills,

Assi,

Hall,

Pokhrel,

Sido,

Keo,

Sodemann,

Davies,

Furbee,

Sparks,

Vincik,

Dedear,

Guo,

Doo,

Burris);

type MonthName is

(January,

February,

March,

April,

May,

June,

July,

August,

September,

October,

November,

December);

type Date is

record

Day : Integer range 1 .. 31;

Month : MonthName;

Year : Integer range 1400 .. 2020;

end record;

type Valid is

(Names,

Dates,

Done);

package ValidIO is new Ada.Text\_IO.Enumeration\_IO(Valid);

use ValidIO;

package NameIO is new Ada.Text\_IO.Enumeration\_IO(Name);

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

type NameArray is array (Integer range <>) of Name;

type DateArray is array (Integer range <>) of Date;

type IntArray is array (Natural range <>) of Integer;

procedure Putname (

Thing : in Name) is

begin

NameIO.Put(Thing);

end;

procedure Putdate (

Thing : in Date) is

begin

MonthIO.Put(Thing.Month);

Put(" ");

Put(Thing.Day, 2);

Put(", ");

Put(Thing.Year, 2);

end;

procedure GetInput (

Input : out Unbounded\_String) is

N : Character;

begin

loop

Get(N);

exit when N = ';';

Append(Input, N);

end loop;

end GetInput;

StackType : Valid;

MemoryStart : Integer;

MemoryEnd : Integer;

Front : Integer;

Rear : Integer;

NumberOfStacks : Integer;

begin

Main:

loop

Put\_Line("Enter a stack type (Names/Dates): ");

Get(StackType);

case StackType is

when Names =>

Put\_Line("Enter the start of memory: ");

Get(MemoryStart);

Put\_Line("Enter the end of memory: ");

Get(MemoryEnd);

Put\_Line("Enter the front of the stack list: ");

Get(Front);

Put\_Line("Enter the rear of the stack list: ");

Get(Rear);

Put\_Line("Enter number of stacks: ");

Get(NumberOfStacks);

declare

procedure MS is

new MoveStacks(Name, NameArray, IntArray);

procedure ReAll is

new Reallocate(Name, NameArray, IntArray, MS);

package NameStacks is new Dynamicstacks(Name, NameArray, IntArray, MemoryStart, MemoryEnd, Front, Rear, NumberOfStacks, 0.09, 0.1, ReAll, Putname);

procedure ProcessNameInput (

A\_String : in Unbounded\_String) is

Temp : String (1 .. Length (A\_String)) := To\_String (A\_String);

Index : Integer := 2;

Stack : Integer;

A\_Name : Name;

Successful : Boolean;

begin

case Temp(1) is

when 'I' =>

while Temp(Index) /= ' ' loop

Index := Index + 1;

end loop;

Stack := Integer'Value(Temp(2..Index-1));

A\_Name := Name'Value(Temp(Index+1..Temp'Last));

NameStacks.Push(Stack, A\_Name, Successful);

if Successful then

Put("Inserted -");

NameIO.Put(A\_Name);

Put("- into stack ");

Put(Stack, 2);

Put\_Line(".");

else

Put(">>Failed to insert ");

NameIO.Put(A\_Name);

Put(" into stack ");

Put(Stack, 1);

Put\_Line("<<");

end if;

when 'D' =>

Stack := Integer'Value(Temp(2..Temp'Last));

NameStacks.Pop(Stack, A\_Name, Successful);

if Successful then

Put("Pop Results for stack ");

Put(Stack, 2);

Put(": ");

NameIO.Put(A\_Name);

New\_Line;

else

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

end if;

when others =>

null;

end case;

end ProcessNameInput;

begin

NameStacks.Setup;

NameLoop:

loop

declare

Temp : Unbounded\_String;

begin

GetInput(Temp);

exit when Temp = "exit";

ProcessNameInput(Temp);

end;

end loop NameLoop;

end;

when Dates=>

Put\_Line("Enter the start of memory: ");

Get(MemoryStart);

Put\_Line("Enter the end of memory: ");

Get(MemoryEnd);

Put\_Line("Enter the front of the stack list: ");

Get(Front);

Put\_Line("Enter the rear of the stack list: ");

Get(Rear);

Put\_Line("Enter number of stacks: ");

Get(NumberOfStacks);

declare

procedure MS is

new MoveStacks(Date, DateArray, IntArray);

procedure ReAll is

new Reallocate(Date, DateArray, IntArray, MS);

package DateStacks is new Dynamicstacks(Date, DateArray, IntArray, MemoryStart, MemoryEnd, Front, Rear, NumberOfStacks, 0.09, 0.1, ReAll, Putdate);

procedure ProcessDateInput (

A\_String : in Unbounded\_String) is

Temp : String (1 .. Length (A\_String)) := To\_String (A\_String);

Index1 : Integer := 2;

Index2 : Integer;

Stack : Integer;

A\_Date : Date;

Successful : Boolean;

begin

case Temp(1) is

when 'I' =>

while Temp(Index1) /= ' ' loop

Index1 := Index1 + 1;

end loop;

Stack := Integer'Value(Temp(2..Index1 - 1));

Index1 := Index1 + 1;

Index2 := Index1;

while Temp(Index1) /= ' ' loop

Index1 := Index1 + 1;

end loop;

A\_Date.Month := MonthName'Value(Temp(Index2..Index1 - 1));

Index1 := Index1 + 1;

Index2 := Index1;

while Temp(Index1) /= ' ' loop

Index1 := Index1 + 1;

end loop;

A\_Date.Day := Integer'Value(Temp(Index2..Index1 - 1));

A\_Date.Year := Integer'Value(Temp(Index1 + 1..Temp'Last));

DateStacks.Push(Stack, A\_Date, Successful);

if Successful then

Put("Inserted -");

Putdate(A\_Date);

Put("- into stack ");

Put(Stack, 2);

Put\_Line(".");

else

Put(">>Failed to insert ");

Putdate(A\_Date);

Put(" into stack ");

Put(Stack, 1);

Put\_Line("<<");

end if;

when 'D' =>

Stack := Integer'Value(Temp(2..Temp'Last));

DateStacks.Pop(Stack, A\_Date, Successful);

if Successful then

Put("Pop Results for stack ");

Put(Stack, 2);

Put(": ");

Putdate(A\_Date);

New\_Line;

else

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

end if;

when others =>

null;

end case;

end ProcessDateInput;

begin

DateStacks.Setup;

DateLoop:

loop

declare

Temp : Unbounded\_String;

begin

GetInput(Temp);

exit when Temp = "exit";

ProcessDateInput(Temp);

end;

end loop DateLoop;

end;

when Done =>

exit Main;

end case;

end loop Main;

end;

**DynamicStacks.ads:**

generic

type ITEM is private;

type ITEM\_ARRAY is array (Integer range <>) of ITEM;

type INT\_ARRAY is array (Natural range <>) of Integer;

Lo : Integer;

M : Integer;

Front : Integer;

Rear : Integer;

NumberOfStacks : Integer;

EqualAllocate : Float;

Minimum : Float;

with procedure Reallocate (

Top : in out INT\_ARRAY;

OldTop : in out INT\_ARRAY;

Base : in out INT\_ARRAY;

StackSpace : in out ITEM\_ARRAY;

MinimumSpace : in Float;

EqualAllocate : in Float;

StackNum : in Integer;

Thing : in ITEM;

Successful : out Boolean);

with procedure Put (

Thing : ITEM);

package DynamicStacks is

procedure Setup;

procedure Push (

K : in Integer;

Thing : in ITEM;

Successful : out Boolean);

procedure Pop (

K : in Integer;

Thing : out ITEM;

Successful : out Boolean);

end DynamicStacks;

**DynamicStacks.adb:**

with Ada.Text\_IO;

use Ada.Text\_IO;

with Ada.Integer\_Text\_IO;

use Ada.Integer\_Text\_IO;

package body DynamicStacks is

Top : INT\_ARRAY (1 .. NumberOfStacks);

Base : INT\_ARRAY (1 .. NumberOfStacks + 1);

StackSpace : ITEM\_ARRAY (Lo..M);

OldTop : INT\_ARRAY (1 .. NumberOfStacks + 1);

procedure Setup is

Total : Integer;

begin

Total := Rear - Front;

for J in 1..NumberOfStacks loop

Base(J) := Integer(Float'Floor(Float(J-1) / Float(NumberOfStacks) \* Float(Total)) + Float(Front));

Top(J) := Base(J);

OldTop(J) := Top(J);

end loop;

Base(NumberOfStacks+1) := Rear;

end;

procedure PrintArrays (

PrintOldTop : in Boolean) is

begin

Put("Base [");

for X in 1..NumberOfStacks+1 loop

Put(Base(X), 2);

if X /= NumberOfStacks+1 then

Put(", ");

end if;

end loop;

Put\_Line("]");

--print top

Put("Top [");

for X in 1..NumberOfStacks loop

Put(Top(X), 2);

if X /= NumberOfStacks then

Put(", ");

end if;

end loop;

Put\_Line("]");

if PrintOldTop then

--print oldtop

Put("OldTop [");

for X in 1..NumberOfStacks loop

Put(OldTop(X), 2);

if X /= NumberOfStacks then

Put(", ");

end if;

end loop;

Put\_Line("]");

end if;

end PrintArrays;

procedure PrintStacks is

begin

for J in 1..NumberOfStacks loop

Put("Stack ");

Put(J, 2);

Put(" [");

for X in Base(J) + 1..Top(J) loop

Put(X, 2);

Put(": ");

Put(StackSpace(X));

if X /= Top(J) then

Put(", ");

end if;

end loop;

Put\_Line("]");

end loop;

end PrintStacks;

procedure Push (

K : in Integer;

Thing : in ITEM;

Successful : out Boolean) is

Temp : Boolean;

begin

Top(K) := Top(K) + 1;

if Top(K) > Base(K+1) then

New\_Line;

Put("OVERFLOW IN STACK ");

Put(K, 2);

Put\_Line("!");

Top(K) := Top(K) - 1;

PrintArrays(True);

PrintStacks;

Top(K) := Top(K) + 1;

Put\_Line("REALLOCATING...");

--Reallocate

Reallocate(Top, OldTop, Base, StackSpace, Minimum, EqualAllocate, K, Thing, Temp);

if Temp then

PrintArrays(False);

PrintStacks;

New\_Line;

Successful := True;

else

Put\_Line("Could Not Reallocate: Not Enough Space.");

New\_Line;

Successful := False;

Top(K) := Top(K) - 1;

end if;

else

StackSpace(Top(K)) := Thing;

Successful := True;

end if;

end;

procedure Pop (

K : in Integer;

Thing : out ITEM;

Successful : out Boolean) is

begin

if Top(K) = Base(K) then

--underflow

Successful := False;

return;

else

Thing := StackSpace(Top(K));

Top(K) := Top(K) - 1;

Successful := True;

end if;

end;

end DynamicStacks;

**MoveStacks.ads:**

generic

type ITEM is private;

type ITEM\_ARRAY is array (Integer range <>) of ITEM;

type INT\_ARRAY is array (Natural range <>) of Integer;

procedure MoveStacks (

Base : in out INT\_ARRAY;

Top : in out INT\_ARRAY;

StackSpace : in out ITEM\_ARRAY;

NewBase : in INT\_ARRAY);

**MoveStacks.adb:**

procedure MoveStacks (

Base : in out INT\_ARRAY;

Top : in out INT\_ARRAY;

StackSpace : in out ITEM\_ARRAY;

NewBase : in INT\_ARRAY) is

Delt : Integer;

N : Integer := Top'Length;

begin

for H in 2..N loop

if NewBase(H) < Base(H) then

Delt := Base(H) - NewBase(H);

for L in (Base(H) +1)..Top(H) loop

StackSpace(L - Delt) := StackSpace(L);

end loop;

Base(H) := NewBase(H);

Top(H) := Top(H) - Delt;

end if;

end loop;

for H in reverse 2..N loop

if NewBase(H) > Base(H) then

Delt := NewBase(H) - Base(H);

for L in reverse (Base(H) + 1)..Top(H) loop

StackSpace(L + Delt) := StackSpace(L);

end loop;

Base(H) := NewBase(H);

Top(H) := Top(H) + Delt;

end if;

end loop;

end MoveStacks;

**Reallocate.ads:**

generic

type ITEM is private;

type ITEM\_ARRAY is array (Integer range <>) of ITEM;

type INT\_ARRAY is array (Natural range <>) of Integer;

with procedure MoveStacks (

Base : in out INT\_ARRAY;

Top : in out INT\_ARRAY;

StackSpace : in out ITEM\_ARRAY;

NewBase : in INT\_ARRAY);

procedure Reallocate (

Top : in out INT\_ARRAY;

OldTop : in out INT\_ARRAY;

Base : in out INT\_ARRAY;

StackSpace : in out ITEM\_ARRAY;

MinimumSpace : in Float;

EqualAllocate : in Float;

StackNum : in Integer;

Thing : in ITEM;

Successful : out Boolean);

**Reallocate.adb:**

procedure Reallocate (

Top : in out INT\_ARRAY;

OldTop : in out INT\_ARRAY;

Base : in out INT\_ARRAY;

StackSpace : in out ITEM\_ARRAY;

MinimumSpace : in Float;

EqualAllocate : in Float;

StackNum : in Integer;

Thing : in ITEM;

Successful : out Boolean) is

N : Integer := Top'Length;

AvailSpace : Integer;

Totallnc : Integer;

J : Integer := N;

GrowthAllocate : Float;

Alpha : Float;

Beta : Float;

Tau : Float;

Sigma : Float;

begin

AvailSpace := Base(N+1) - Base(1);

Totallnc := 0;

while J > 0 loop

AvailSpace := AvailSpace - (Top(J) - Base(J));

if Top(J) > OldTop(J) then

OldTop(J+1) := Top(J) - OldTop(J);

Totallnc := Totallnc + OldTop(J+1);

else

OldTop(J+1) := 0;

end if;

J := J - 1;

end loop;

if Float(AvailSpace) < ((Float(Base(N+1) - Base(1)) \* MinimumSpace) - 1.0) then

--not enough memory

Successful := False;

Top(StackNum) := Top(StackNum) - 1;

for X in 1..N loop

OldTop(X) := Top(X); --currently holds growth values

end loop;

Top(StackNum) := Top(StackNum) + 1;

return;

end if;

GrowthAllocate := 1.0 - EqualAllocate;

Alpha := (EqualAllocate \* Float(AvailSpace)) / Float(N);

Beta := (GrowthAllocate \* Float(AvailSpace)) / Float(Totallnc);

OldTop(1) := Base(1);

Sigma := 0.0;

for I in 2..N loop

Tau := Sigma + Alpha + (Float(OldTop(I)) \* Beta);

OldTop(I) := OldTop(I - 1) + (Top(I - 1) - Base(I - 1)) + Integer(Float'Floor(Tau)) - Integer(Float'Floor(Sigma));

Sigma := Tau;

end loop;

Top(StackNum) := Top(StackNum) - 1;

--MoveStack Begin

MoveStacks(Base, Top, StackSpace, OldTop);

--MoveStack End

Top(StackNum) := Top(StackNum) + 1;

StackSpace(Top(StackNum)) := Thing;

for X in 1..N loop

OldTop(X) := Top(X);

end loop;

Successful := True;

end Reallocate;