Christopher Aboshear

Data Structures 3319

4/11/19

Successfully completed “A” option

**Input “C/B”**

37

3

0

-11

60

I1 David

I1 Joe

I1 Bob

I1 Devin

I1 Elwin

I3 Sagar

I3 Ryan

D1

I2 Kyle

I2 Tyler

I2 Minh

I3 Adrian

I3 Daniel

I3 Jailene

I2 Jacob

D1

D1

D1

I2 Bryce

I2 Ryan

I2 Jaylene

I2 Sagar

I2 Ayran

I2 Travis

I2 Devin

D3

D3

I2 Rahul

I2 Elwin

I2 Victor

I3 Corey

I3 Bipin

I3 Frank

I3 Nathan

I3 Amber

I3 Amy

I3 No

I3 Hope

**Output “C/B”**

------------INIT--------------

v= 0.00000000000000E+00

x= 0.00000000000000E+00

fl= 1.00000000000000E+00

top[ 1]= 1

base[ 1]= 1

arr[ 1]= 1

v= 3.33333333333333E-01

x= 1.23333333333333E+01

fl= 1.30000000000000E+01

top[ 2]= 13

base[ 2]= 13

arr[ 2]= 13

v= 6.66666666666667E-01

x= 2.46666666666667E+01

fl= 2.50000000000000E+01

top[ 3]= 25

base[ 3]= 25

arr[ 3]= 25

base[ 4]= 37

------------INIT--------------

--------------INSERT------------

val=DAVID

top[ 1]= 2

base[ 2]= 13

top[ 1]= 2

------------INSERT-------------

--------------INSERT------------

val=JOE

top[ 1]= 3

base[ 2]= 13

top[ 1]= 3

------------INSERT-------------

--------------INSERT------------

val=BOB

top[ 1]= 4

base[ 2]= 13

top[ 1]= 4

------------INSERT-------------

--------------INSERT------------

val=DEVIN

top[ 1]= 5

base[ 2]= 13

top[ 1]= 5

------------INSERT-------------

--------------INSERT------------

val=ELWIN

top[ 1]= 6

base[ 2]= 13

top[ 1]= 6

------------INSERT-------------

--------------INSERT------------

val=SAGAR

top[ 3]= 26

base[ 4]= 37

top[ 3]= 26

------------INSERT-------------

--------------INSERT------------

val=RYAN

top[ 3]= 27

base[ 4]= 37

top[ 3]= 27

------------INSERT-------------

-----------DELETE-----------

temp=ELWIN

top[ 1]= 5

-----------DELETE--------------

--------------INSERT------------

val=KYLE

top[ 2]= 14

base[ 3]= 25

top[ 2]= 14

------------INSERT-------------

--------------INSERT------------

val=TYLER

top[ 2]= 15

base[ 3]= 25

top[ 2]= 15

------------INSERT-------------

--------------INSERT------------

val=MINH

top[ 2]= 16

base[ 3]= 25

top[ 2]= 16

------------INSERT-------------

--------------INSERT------------

val=ADRIAN

top[ 3]= 28

base[ 4]= 37

top[ 3]= 28

------------INSERT-------------

--------------INSERT------------

val=DANIEL

top[ 3]= 29

base[ 4]= 37

top[ 3]= 29

------------INSERT-------------

--------------INSERT------------

val=JAILENE

top[ 3]= 30

base[ 4]= 37

top[ 3]= 30

------------INSERT-------------

--------------INSERT------------

val=JACOB

top[ 2]= 17

base[ 3]= 25

top[ 2]= 17

------------INSERT-------------

-----------DELETE-----------

temp=DEVIN

top[ 1]= 4

-----------DELETE--------------

-----------DELETE-----------

temp=BOB

top[ 1]= 3

-----------DELETE--------------

-----------DELETE-----------

temp=JOE

top[ 1]= 2

-----------DELETE--------------

--------------INSERT------------

val=BRYCE

top[ 2]= 18

base[ 3]= 25

top[ 2]= 18

------------INSERT-------------

--------------INSERT------------

val=RYAN

top[ 2]= 19

base[ 3]= 25

top[ 2]= 19

------------INSERT-------------

--------------INSERT------------

val=JAYLENE

top[ 2]= 20

base[ 3]= 25

top[ 2]= 20

------------INSERT-------------

--------------INSERT------------

val=SAGAR

top[ 2]= 21

base[ 3]= 25

top[ 2]= 21

------------INSERT-------------

--------------INSERT------------

val=AYRAN

top[ 2]= 22

base[ 3]= 25

top[ 2]= 22

------------INSERT-------------

--------------INSERT------------

val=TRAVIS

top[ 2]= 23

base[ 3]= 25

top[ 2]= 23

------------INSERT-------------

--------------INSERT------------

val=DEVIN

top[ 2]= 24

base[ 3]= 25

top[ 2]= 24

------------INSERT-------------

-----------DELETE-----------

temp=JAILENE

top[ 3]= 29

-----------DELETE--------------

-----------DELETE-----------

temp=DANIEL

top[ 3]= 28

-----------DELETE--------------

--------------INSERT------------

val=RAHUL

top[ 2]= 25

base[ 3]= 25

top[ 2]= 25

------------INSERT-------------

--------------INSERT------------

val=ELWIN

top[ 2]= 26

base[ 3]= 25

Overflow occured

-------------REALLOC---------------

top[ 2]= 26

base[ 2]= 13

old\_top[ 2]= 13

top[ 2]= 26

base[ 2]= 13

old\_top[ 2]= 13

top[ 2]= 26

base[ 2]= 13

old\_top[ 2]= 13

-------------STEP 1----------------

avail\_space= 37

total\_inc= 0

j= 3

------------J= 3-------------

top[ 3]= 28

base[ 3]= 25

old\_top[ 3]= 25

total\_inc= 3

avail\_space= 34

growth[ 3]= 25

----------------------J= 3---------------

------------J= 2-------------

top[ 2]= 26

base[ 2]= 13

old\_top[ 2]= 13

total\_inc= 16

avail\_space= 21

growth[ 2]= 13

----------------------J= 2---------------

------------J= 1-------------

top[ 1]= 2

base[ 1]= 1

old\_top[ 1]= 1

total\_inc= 17

avail\_space= 20

growth[ 1]= 1

----------------------J= 1---------------

total\_inc= 17

avail\_space= 20

min\_space= 1

---------------STEP 1---------------

---------------STEP 2---------------

---------------STEP 2---------------

---------------STEP 3---------------

equal\_alloc= 2.30000000000000E-01

growth\_alloc= 7.70000000000000E-01

alpha= 1.53333333333333E+00

---------------STEP 3---------------

---------------STEP 4---------------

beta= 9.05882352941176E-01

---------------STEP 4---------------

---------------STEP 5---------------

arr[1]= 1

sigma= 0.00000000000000E+00

--------------------J= 2----------

tau= 2.43921568627451E+00

new\_base[ 2]= 4

sigma= 2.43921568627451E+00

--------------------J= 2----------

--------------------J= 3----------

tau= 1.57490196078431E+01

new\_base[ 3]= 30

sigma= 1.57490196078431E+01

--------------------J= 3----------

---------------STEP 5---------------

---------------STEP 6---------------

top[ 2]= 25

------------------------MOVE STACK------------------

-----------------STEP 1------------------

-----------------J= 2------------------

Arr[ 2]= 4

base[ 2]= 13

delta= 9

base[ 2]= 13

top[ 2]= 25

--------------------------L= 14-----------------------

-----------------J= 2------------------

-----------------J= 3------------------

Arr[ 3]= 30

base[ 3]= 25

-----------------J= 3------------------

-----------------STEP 1------------------

-----------------STEP 2------------------

-----------------J= 2------------------

Arr[ 2]= 4

base[ 2]= 4

-----------------J= 2------------------

-----------------J= 3------------------

Arr[ 3]= 30

base[ 3]= 25

delta= 5

base[ 3]= 25

top[ 3]= 28

--------------------------L= 28-----------------------

Stack\_space[ 23]=ADRIAN

--------------------------L= 27-----------------------

--------------------------L= 27-----------------------

Stack\_space[ 22]=RYAN

--------------------------L= 26-----------------------

--------------------------L= 26-----------------------

Stack\_space[ 21]=SAGAR

--------------------------L= 25-----------------------

--------------------------L= 25-----------------------

-----------------J= 3------------------

-----------------STEP 2------------------

------------------------MOVE STACK------------------

top[ 2]= 17

--------------INSERT------------

val=ELWIN

top[ 2]= 18

base[ 3]= 30

top[ 2]= 18

------------INSERT-------------

Arr[ 1]= 2

Arr[ 2]= 18

Arr[ 3]= 33

---------------STEP 6---------------

-------------REALLOC---------------

------------INSERT-------------

--------------INSERT------------

val=VICTOR

top[ 2]= 19

base[ 3]= 30

top[ 2]= 19

------------INSERT-------------

--------------INSERT------------

val=COREY

top[ 3]= 34

base[ 4]= 37

top[ 3]= 34

------------INSERT-------------

--------------INSERT------------

val=BIPIN

top[ 3]= 35

base[ 4]= 37

top[ 3]= 35

------------INSERT-------------

--------------INSERT------------

val=FRANK

top[ 3]= 36

base[ 4]= 37

top[ 3]= 36

------------INSERT-------------

--------------INSERT------------

val=NATHAN

top[ 3]= 37

base[ 4]= 37

top[ 3]= 37

------------INSERT-------------

--------------INSERT------------

val=AMBER

top[ 3]= 38

base[ 4]= 37

Overflow occured

-------------REALLOC---------------

top[ 3]= 38

base[ 3]= 30

old\_top[ 3]= 33

top[ 3]= 38

base[ 3]= 30

old\_top[ 3]= 33

top[ 3]= 38

base[ 3]= 30

old\_top[ 3]= 33

-------------STEP 1----------------

avail\_space= 37

total\_inc= 0

j= 3

------------J= 3-------------

top[ 3]= 38

base[ 3]= 30

old\_top[ 3]= 33

total\_inc= 5

avail\_space= 29

growth[ 3]= 33

----------------------J= 3---------------

------------J= 2-------------

top[ 2]= 19

base[ 2]= 4

old\_top[ 2]= 18

total\_inc= 6

avail\_space= 14

growth[ 2]= 18

----------------------J= 2---------------

------------J= 1-------------

top[ 1]= 2

base[ 1]= 1

old\_top[ 1]= 2

total\_inc= 6

avail\_space= 13

growth[ 1]= 2

----------------------J= 1---------------

total\_inc= 6

avail\_space= 13

min\_space= 1

---------------STEP 1---------------

---------------STEP 2---------------

---------------STEP 2---------------

---------------STEP 3---------------

equal\_alloc= 2.30000000000000E-01

growth\_alloc= 7.70000000000000E-01

alpha= 9.96666666666667E-01

---------------STEP 3---------------

---------------STEP 4---------------

beta= 1.66833333333333E+00

---------------STEP 4---------------

---------------STEP 5---------------

arr[1]= 1

sigma= 0.00000000000000E+00

--------------------J= 2----------

tau= 9.96666666666667E-01

new\_base[ 2]= 2

sigma= 9.96666666666667E-01

--------------------J= 2----------

--------------------J= 3----------

tau= 3.66166666666667E+00

new\_base[ 3]= 20

sigma= 3.66166666666667E+00

--------------------J= 3----------

---------------STEP 5---------------

---------------STEP 6---------------

top[ 3]= 37

------------------------MOVE STACK------------------

-----------------STEP 1------------------

-----------------J= 2------------------

Arr[ 2]= 2

base[ 2]= 4

delta= 2

base[ 2]= 4

top[ 2]= 19

--------------------------L= 5-----------------------

-----------------J= 2------------------

-----------------J= 3------------------

Arr[ 3]= 20

base[ 3]= 30

delta= 10

base[ 3]= 30

top[ 3]= 37

--------------------------L= 31-----------------------

-----------------J= 3------------------

-----------------STEP 1------------------

-----------------STEP 2------------------

-----------------J= 2------------------

Arr[ 2]= 2

base[ 2]= 2

-----------------J= 2------------------

-----------------J= 3------------------

Arr[ 3]= 20

base[ 3]= 20

-----------------J= 3------------------

-----------------STEP 2------------------

------------------------MOVE STACK------------------

top[ 3]= 28

--------------INSERT------------

val=AMBER

top[ 3]= 29

base[ 4]= 37

top[ 3]= 29

------------INSERT-------------

Arr[ 1]= 2

Arr[ 2]= 17

Arr[ 3]= 29

---------------STEP 6---------------

-------------REALLOC---------------

------------INSERT-------------

--------------INSERT------------

val=AMY

top[ 3]= 30

base[ 4]= 37

top[ 3]= 30

------------INSERT-------------

--------------INSERT------------

val=NO

top[ 3]= 31

base[ 4]= 37

top[ 3]= 31

------------INSERT-------------

--------------INSERT------------

val=HOPE

top[ 3]= 32

base[ 4]= 37

top[ 3]= 32

------------INSERT-------------

**Input “A”**

13

3

4

0

50

I2 January 15 1956

I2 February 14 1957

I3 September 16 1946

I2 September 17 1842

I2 April 1 2015

I1 December 24 1996

D1

I3 March 16 1992

D1

I2 January 15 1956

I3 April 4 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

**Output “A”**

Please Enter Input File:

input\_multistack\_a.txt

------------INIT--------------

v= 0.00000000000000E+00

x= 0.00000000000000E+00

fl= 4.00000000000000E+00

top[ 1]= 4

base[ 1]= 4

arr[ 1]= 4

v= 3.33333333333333E-01

x= 4.33333333333333E+00

fl= 8.00000000000000E+00

top[ 2]= 8

base[ 2]= 8

arr[ 2]= 8

v= 6.66666666666667E-01

x= 8.66666666666667E+00

fl= 1.20000000000000E+01

top[ 3]= 12

base[ 3]= 12

arr[ 3]= 12

base[ 4]= 17

------------INIT--------------

--------------INSERT------------

val= 1956 JANUARY 15

top[ 2]= 9

base[ 3]= 12

top[ 2]= 9

------------INSERT-------------

--------------INSERT------------

val= 1957 FEBRUARY 14

top[ 2]= 10

base[ 3]= 12

top[ 2]= 10

------------INSERT-------------

--------------INSERT------------

val= 1946 SEPTEMBER 16

top[ 3]= 13

base[ 4]= 17

top[ 3]= 13

------------INSERT-------------

--------------INSERT------------

val= 1842 SEPTEMBER 17

top[ 2]= 11

base[ 3]= 12

top[ 2]= 11

------------INSERT-------------

--------------INSERT------------

val= 2015 APRIL 1

top[ 2]= 12

base[ 3]= 12

top[ 2]= 12

------------INSERT-------------

--------------INSERT------------

val= 1996 DECEMBER 24

top[ 1]= 5

base[ 2]= 8

top[ 1]= 5

------------INSERT-------------

-----------DELETE-----------

temp= 1996 DECEMBER 24

top[ 1]= 4

-----------DELETE--------------

--------------INSERT------------

val= 1992 MARCH 16

top[ 3]= 14

base[ 4]= 17

top[ 3]= 14

------------INSERT-------------

-----------DELETE-----------

------------UNDERFLOW--------

Underflow

------------UNDERFLOW--------

-----------DELETE--------------

--------------INSERT------------

val= 1956 JANUARY 15

top[ 2]= 13

base[ 3]= 12

Overflow occured

-------------REALLOC---------------

top[ 2]= 13

base[ 2]= 8

old\_top[ 2]= 8

top[ 2]= 13

base[ 2]= 8

old\_top[ 2]= 8

top[ 2]= 13

base[ 2]= 8

old\_top[ 2]= 8

-------------STEP 1----------------

avail\_space= 9

total\_inc= 0

j= 3

------------J= 3-------------

top[ 3]= 14

base[ 3]= 12

old\_top[ 3]= 12

total\_inc= 2

avail\_space= 7

growth[ 3]= 12

----------------------J= 3---------------

------------J= 2-------------

top[ 2]= 13

base[ 2]= 8

old\_top[ 2]= 8

total\_inc= 7

avail\_space= 2

growth[ 2]= 8

----------------------J= 2---------------

------------J= 1-------------

top[ 1]= 4

base[ 1]= 4

old\_top[ 1]= 4

total\_inc= 7

avail\_space= 2

growth[ 1]= 4

----------------------J= 1---------------

total\_inc= 7

avail\_space= 2

min\_space= 0

---------------STEP 1---------------

---------------STEP 2---------------

---------------STEP 2---------------

---------------STEP 3---------------

equal\_alloc= 2.30000000000000E-01

growth\_alloc= 7.70000000000000E-01

alpha= 1.53333333333333E-01

---------------STEP 3---------------

---------------STEP 4---------------

beta= 2.20000000000000E-01

---------------STEP 4---------------

---------------STEP 5---------------

arr[1]= 4

sigma= 0.00000000000000E+00

--------------------J= 2----------

tau= 1.53333333333333E-01

new\_base[ 2]= 4

sigma= 1.53333333333333E-01

--------------------J= 2----------

--------------------J= 3----------

tau= 1.40666666666667E+00

new\_base[ 3]= 10

sigma= 1.40666666666667E+00

--------------------J= 3----------

---------------STEP 5---------------

---------------STEP 6---------------

top[ 2]= 12

------------------------MOVE STACK------------------

-----------------STEP 1------------------

-----------------J= 2------------------

Arr[ 2]= 4

base[ 2]= 8

delta= 4

base[ 2]= 8

top[ 2]= 12

--------------------------L= 9-----------------------

-----------------J= 2------------------

-----------------J= 3------------------

Arr[ 3]= 10

base[ 3]= 12

delta= 2

base[ 3]= 12

top[ 3]= 14

--------------------------L= 13-----------------------

-----------------J= 3------------------

-----------------STEP 1------------------

-----------------STEP 2------------------

-----------------J= 2------------------

Arr[ 2]= 4

base[ 2]= 4

-----------------J= 2------------------

-----------------J= 3------------------

Arr[ 3]= 10

base[ 3]= 10

-----------------J= 3------------------

-----------------STEP 2------------------

------------------------MOVE STACK------------------

top[ 2]= 9

--------------INSERT------------

val= 1956 JANUARY 15

top[ 2]= 10

base[ 3]= 10

top[ 2]= 10

------------INSERT-------------

Arr[ 1]= 4

Arr[ 2]= 10

Arr[ 3]= 12

---------------STEP 6---------------

-------------REALLOC---------------

------------INSERT-------------

--------------INSERT------------

val= 1492 APRIL 4

top[ 3]= 13

base[ 4]= 17

top[ 3]= 13

------------INSERT-------------

--------------INSERT------------

val= 1776 NOVEMBER 7

top[ 3]= 14

base[ 4]= 17

top[ 3]= 14

------------INSERT-------------

--------------INSERT------------

val= 1994 JUNE 12

top[ 3]= 15

base[ 4]= 17

top[ 3]= 15

------------INSERT-------------

--------------INSERT------------

val= 1776 JULY 4

top[ 2]= 11

base[ 3]= 10

Overflow occured

-------------REALLOC---------------

top[ 2]= 11

base[ 2]= 4

old\_top[ 2]= 10

top[ 2]= 11

base[ 2]= 4

old\_top[ 2]= 10

top[ 2]= 11

base[ 2]= 4

old\_top[ 2]= 10

-------------STEP 1----------------

avail\_space= 9

total\_inc= 0

j= 3

------------J= 3-------------

top[ 3]= 15

base[ 3]= 10

old\_top[ 3]= 12

total\_inc= 3

avail\_space= 4

growth[ 3]= 12

----------------------J= 3---------------

------------J= 2-------------

top[ 2]= 11

base[ 2]= 4

old\_top[ 2]= 10

total\_inc= 4

avail\_space=-3

growth[ 2]= 10

----------------------J= 2---------------

------------J= 1-------------

top[ 1]= 4

base[ 1]= 4

old\_top[ 1]= 4

total\_inc= 4

avail\_space=-3

growth[ 1]= 4

----------------------J= 1---------------

total\_inc= 4

avail\_space=-3

min\_space= 0

---------------STEP 1---------------

---------------STEP 2---------------

Out of Memory

------------INSERT-------------

**Multistack.ads**

with Ada.Text\_IO;

use Ada.Text\_IO;

--Generic instantiation

generic

--item is private type

type item is private;

with function image(i1: in item) return String;

--package = class in java or C++

package multistack is

--Declared array type integer

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

--Declared array of type item

type item\_arr is array (Integer range <>) of item;

--Move stack variables instantiated

procedure move\_stack(

base : in out Int\_Arr;

stack\_space : in out item\_arr;

top : in out Int\_Arr;

n : in Integer;

arr : in Int\_Arr

);

--Reallocation variables instantiated

function realloc(

base : in out Int\_Arr;

top : in out Int\_Arr;

arr : in out Int\_Arr;

stack\_space : in out Item\_arr;

m : in Integer;

l0 : in Integer;

n : in Integer;

k : in Integer;

val : in item

) return Boolean;

procedure init\_stack(

base : in out Int\_Arr;

top : in out Int\_Arr;

arr : in out Int\_Arr;

size : in Integer;

tot\_size : in Integer;

l0 : in Integer

);

--Insert stack variables instantiated

function insert\_stack(

base : in out Int\_Arr;

top : in out Int\_Arr;

arr : in out Int\_Arr;

m : in Integer;

l0 : in Integer;

n : in Integer;

k : in Integer;

stack\_space : in out item\_arr;

val : in item

) return Boolean;

--Delete stack variables instantiated

procedure delete\_stack(

base : in Int\_Arr;

top : in out Int\_Arr;

k : in Integer;

stack\_space : in item\_arr

);

end multistack;

**Multistack.adb**

package body multistack is

--Initialization of varbles for deletion in multistack

procedure delete\_stack(

base : in Int\_Arr;

top : in out Int\_Arr;

k : in Integer;

stack\_space : in item\_arr

) is

begin

Ada.Text\_IO.Put\_Line("-----------DELETE-----------");

--Condition if all the stacks are empty

--Print Underflow to the screen

if top(k) = base(k) then

Ada.Text\_IO.Put\_Line("------------UNDERFLOW--------");

Ada.Text\_IO.Put\_Line("Underflow");

Ada.Text\_IO.Put\_Line("------------UNDERFLOW--------");

Ada.Text\_IO.Put\_Line("-----------DELETE--------------");

--Condition when at least one spot is filled

else

--decrement top

--print temp=, top[]=, DELETE to screen

top(k) := top(k) - 1;

Ada.Text\_IO.Put\_Line("temp=" & image(stack\_space(top(k)+1)));

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(k) & "]=" & Integer'Image(top(k)));

Ada.Text\_IO.Put\_Line("-----------DELETE--------------");

end if;

end delete\_stack;

--Initialization of variables for insertion in multistack

function insert\_stack(

base : in out Int\_Arr;

top : in out Int\_Arr;

arr : in out Int\_Arr;

m : in Integer;

l0 : in Integer;

n : in Integer;

k : in Integer;

stack\_space : in out item\_arr;

val : in item

) return Boolean is

temp\_boolean : Boolean;

--Whats printed to screen for insertion function

begin

--Print to screen

Ada.Text\_IO.Put\_Line("--------------INSERT------------");

--Print to screen

Ada.Text\_IO.Put\_Line("val=" & image(val));

--Increment top

top(k) := top(k) + 1;

--Print to screen

Ada.Text\_IO.Put\_Line("top["& Integer'Image(k) &"]=" & Integer'Image(top(k)));

--Print to screen

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(k + 1) & "]=" & Integer'Image(base(k+1)));

--Condition if all stacks are full

if top(k) > base(k + 1) then

--Print Overflow

Ada.Text\_IO.Put\_Line("Overflow occured");

--reallocation variables

temp\_boolean := realloc(

base,

top,

arr,

stack\_space,

m,

l0,

n,

k,

val

);

--Print to screen

Ada.Text\_IO.Put\_Line("------------INSERT-------------");

return temp\_boolean;

--Print the available stack space

else

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(k) & "]=" & Integer'Image(top(k)));

stack\_space(top(k)):=val;

Ada.Text\_IO.Put\_Line("------------INSERT-------------");

return True;

end if;

end insert\_stack;

--Initialization of stack

procedure init\_stack(

base : in out Int\_Arr;

top : in out Int\_Arr;

arr : in out Int\_Arr;

size : in Integer;

tot\_size : in Integer;

l0 : in Integer

) is

begin

Ada.Text\_IO.Put\_Line("------------INIT--------------");

--iterate over stacks

for i in Integer range 1..size loop

--Variables for size of stacks and total

declare

v : Long\_Float := Long\_Float(i-1) / Long\_Float(size);

x : Long\_Float := v \* Long\_Float(tot\_size);

fl : Long\_Float := Long\_Float'Floor(x) + Long\_Float(l0);

begin

Ada.Text\_IO.Put\_Line("v=" & Long\_Float'Image(v));

Ada.Text\_IO.Put\_Line("x=" & Long\_Float'Image(x));

Ada.Text\_IO.Put\_Line("fl=" & Long\_Float'Image(fl));

--Initialize base and top to same location at start

top(i) := Integer(fl);

base(i) := top(i);

arr(i) := top(i);

--Print top, base, and arr

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(i) & "]=" & Integer'Image(top(i)));

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(i) & "]=" & Integer'Image(base(i)));

Ada.Text\_IO.Put\_Line("arr[" & Integer'Image(i) & "]=" & Integer'Image(arr(i)));

end;

end loop;

--Declare variable of type long\_float, takes the floor of numbers

declare

v : Long\_Float := Long\_Float'Floor(Long\_Float(tot\_size)) + Long\_Float(l0);

begin

--increment base

base(size + 1) := Integer(v);

end;

--Print base to screen

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(size+1) & "]=" & Integer'Image(base(size+1)));

Ada.TExt\_IO.Put\_Line("------------INIT--------------");

end init\_stack;

--Variables in move stack declared

procedure move\_stack(

base : in out Int\_Arr;

stack\_space : in out item\_arr;

top : in out Int\_Arr;

n : in Integer;

arr : in Int\_Arr

) is

begin

Ada.Text\_IO.Put\_Line("------------------------MOVE STACK------------------");

Ada.Text\_IO.Put\_Line("-----------------STEP 1------------------");

--Iterate and find out if anything is in the stack

for j in Integer range 2..n loop

--print to the screen what is found

Ada.Text\_IO.Put\_Line("-----------------J=" & Integer'Image(j) &"------------------");

Ada.Text\_IO.Put\_Line("Arr[" & Integer'Image(j) & "]=" & Integer'Image(arr(j)));

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(j) & "]=" & Integer'Image(base(j)));

--condition when arr is less than base

if arr(j) < base(j) then

--Declare block for delt, counter , and l

declare

delt : Integer := base(j) - arr(j);

counter : Integer := 1;

l : Integer := base(j) + counter;

--Print to screen results of delta, base, top

begin

Ada.Text\_IO.Put\_Line("delta=" & Integer'Image(delt));

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(j) & "]=" & Integer'Image(base(j)));

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(j) & "]=" & Integer'Image(top(j)));

loop

Ada.Text\_IO.Put\_Line("--------------------------L=" & Integer'Image(l) & "-----------------------");

--is statment for when l kis less than top

if l < top(j) then

exit;

end if;

--Calculations for stack space avail

stack\_space(l-delt) := stack\_space(l);

Ada.Text\_IO.Put\_Line("Stack\_space[" & Integer'Image(l-delt) & "]=" & image(stack\_space(l)));

counter := counter + 1;

l := base(j) + counter;

Ada.Text\_IO.Put\_Line("--------------------------L=" & Integer'Image(l) & "-----------------------");

end loop;

--Set base to arr

base(j) := arr(j);

top(j) := top(j) - delt;

end;

end if;

Ada.Text\_IO.Put\_Line("-----------------J=" & Integer'Image(j) &"------------------");

end loop;

Ada.Text\_IO.Put\_Line("-----------------STEP 1------------------");

Ada.Text\_IO.Put\_Line("-----------------STEP 2------------------");

--Iterate over stacks

for j in Integer range 2..n loop

--Print to screen current stack

Ada.Text\_IO.Put\_Line("-----------------J=" & Integer'Image(j) &"------------------");

Ada.Text\_IO.Put\_Line("Arr[" & Integer'Image(j) & "]=" & Integer'Image(arr(j)));

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(j) & "]=" & Integer'Image(base(j)));

--Condition when arr(j) is greater than base (j)

if arr(j) > base(j) then

--Declare block to instantiate delt, counter , and l

declare

delt : Integer := arr(j) - base(j);

counter : Integer := 0;

l : Integer := top(j) - counter;

begin

--Print to the screen current delta, base , top

Ada.Text\_IO.Put\_Line("delta=" & Integer'Image(delt));

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(j) & "]=" & Integer'Image(base(j)));

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(j) & "]=" & Integer'Image(top(j)));

loop

Ada.Text\_IO.Put\_Line("--------------------------L=" & Integer'Image(l) & "-----------------------");

--Condition for l less than base(j)

if l < base(j) + 1 then

exit;

end if;

--New stack space avail

stack\_space(l+delt) := stack\_space(l);

--Print avail stack space

Ada.Text\_IO.Put\_Line("Stack\_space[" & Integer'Image(l-delt) & "]=" & image(stack\_space(l)));

--increment counter

counter := counter + 1;

l := top(j) - counter;

Ada.Text\_IO.Put\_Line("--------------------------L=" & Integer'Image(l) & "-----------------------");

end loop;

--Set base(j) to arr(j)

base(j) := arr(j);

top(j) := top(j) + delt;

end;

end if;

Ada.Text\_IO.Put\_Line("-----------------J=" & Integer'Image(j) &"------------------");

end loop;

Ada.Text\_IO.Put\_Line("-----------------STEP 2------------------");

Ada.Text\_IO.Put\_Line("------------------------MOVE STACK------------------");

end move\_stack;

--Variables in realloc instantiated

function realloc(

base : in out Int\_Arr;

top : in out Int\_Arr;

arr : in out Int\_Arr;

stack\_space : in out Item\_arr;

m : in Integer;

l0 : in Integer;

n : in Integer;

k : in Integer;

val : in item

) return Boolean is

--Variables to account for changes made to stacks are

-- all being instantiated below

avail\_space : Integer := m - l0;

total\_inc : Integer := 0;

min\_space : Integer := Integer(Float'Floor(Float(m) \* 0.05));

j : Integer := n;

growth\_alloc : Long\_Float := 1.0 - 0.23;

equal\_alloc : Long\_Float := 0.23;

alpha : Long\_Float;

beta : Long\_Float;

sigma : Long\_Float;

tau : Long\_Float;

temp\_boolean : Boolean;

begin

Ada.Text\_IO.Put\_Line("-------------REALLOC---------------");

--Iterate over the stacks

for i in Integer range 1..n loop

--Print to screen top, base, and old top

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(k) & "]=" & Integer'Image(top(k)));

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(k) & "]=" & Integer'Image(base(k)));

Ada.Text\_IO.Put\_Line("old\_top[" & Integer'Image(k) & "]=" & Integer'Image(arr(k)));

end loop;

--Print to screen avail\_space, total\_inc, and j

Ada.Text\_IO.Put\_Line("-------------STEP 1----------------");

Ada.Text\_IO.Put\_Line("avail\_space=" & Integer'Image(avail\_space));

Ada.Text\_IO.Put\_Line("total\_inc=" & Integer'Image(total\_inc));

Ada.Text\_IO.Put\_Line("j=" & Integer'Image(j));

--Condition check for when j is greater than 0

while j > 0 loop

--Print to screen j, top, base, old\_top

Ada.Text\_IO.Put\_Line("------------J=" & Integer'Image(j) & "-------------");

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(j) & "]=" & Integer'Image(top(j)));

Ada.Text\_IO.Put\_Line("base[" & Integer'Image(j) & "]=" & Integer'Image(base(j)));

Ada.Text\_IO.Put\_Line("old\_top[" & Integer'Image(j) & "]=" & Integer'Image(arr(j)));

avail\_space := avail\_space - (top(j) - base(j));

--Condition check when top(j) is greater than arr(j)

if top(j) > arr(j) then

arr(j+1) := top(j) - arr(j);

--Get new total\_inc

total\_inc := total\_inc + arr(j+1);

else

--else condition set arr(j+1) to 0

arr(j+1) := 0;

end if;

--Print to screen total\_inc, avail\_space, growth, and j

Ada.Text\_IO.Put\_Line("total\_inc=" & Integer'Image(total\_inc));

Ada.Text\_IO.Put\_Line("avail\_space=" & Integer'Image(avail\_space));

Ada.Text\_IO.Put\_Line("growth[" & Integer'Image(j) & "]=" & Integer'Image(arr(j)));

Ada.Text\_IO.Put\_Line("----------------------J=" & Integer'Image(j) & "---------------");

j := j -1;

end loop;

--Print to screen total\_inc, avail\_space, min\_space

Ada.Text\_IO.Put\_Line("total\_inc=" & Integer'Image(total\_inc));

Ada.Text\_IO.Put\_Line("avail\_space=" & Integer'Image(avail\_space));

Ada.Text\_IO.Put\_Line("min\_space=" & Integer'Image(min\_space));

Ada.Text\_IO.Put\_Line("---------------STEP 1---------------");

Ada.Text\_IO.Put\_Line("---------------STEP 2---------------");

--Condition check when available space is greater

-- than minimum space -1

if avail\_space < (min\_space - 1) then

--Print Out of memory to the screen

Ada.Text\_IO.Put\_Line("Out of Memory");

return False;

end if;

Ada.Text\_IO.Put\_Line("---------------STEP 2---------------");

Ada.Text\_IO.Put\_Line("---------------STEP 3---------------");

--Calculation for the variable alpha

alpha := equal\_alloc \* Long\_Float(avail\_space) / Long\_Float(n);

--Print equal\_alloc, growth\_alloc, and alpha to screen

Ada.Text\_IO.Put\_Line("equal\_alloc=" & Long\_Float'Image(equal\_alloc));

Ada.Text\_IO.Put\_Line("growth\_alloc=" & Long\_Float'Image(growth\_alloc));

Ada.Text\_IO.Put\_Line("alpha=" & Long\_Float'Image(alpha));

Ada.Text\_IO.Put\_Line("---------------STEP 3---------------");

Ada.Text\_IO.Put\_Line("---------------STEP 4---------------");

--Calculations for the variable beta

beta := growth\_alloc \* Long\_Float(avail\_space) / Long\_Float(total\_inc);

--Print the variable beta to the screen

Ada.Text\_IO.Put\_Line("beta=" & Long\_Float'Image(beta));

Ada.Text\_IO.Put\_Line("---------------STEP 4---------------");

Ada.Text\_IO.Put\_Line("---------------STEP 5---------------");

--Set arr(1) to base(1)

arr(1) := base(1);

--Set sigma to long float at postion 0

sigma := Long\_Float(0);

--Print arr[1] and sigma to the screen

Ada.Text\_IO.Put\_Line("arr[1]=" & Integer'Image(arr(1)));

Ada.Text\_IO.Put\_Line("sigma=" & Long\_Float'Image(sigma));

--Iterate over the stacks

for j in Integer range 2..n loop

Ada.Text\_IO.Put\_Line("--------------------J=" & Integer'Image(j) & "----------");

--Calculations for the variable tau

tau := sigma + alpha + Long\_Float(arr(j)) \* beta;

arr(j) := Integer(Long\_Float(arr(j-1)) + (Long\_Float(top(j-1)) - Long\_Float(base(j-1))) + Long\_Float'Floor(tau) - Long\_Float'Floor(sigma));

--Set sigma to the variable tau

sigma := tau;

--Print tau, new\_base, sigma, and j to the screen

Ada.Text\_IO.Put\_Line("tau=" & Long\_Float'Image(tau));

Ada.Text\_IO.Put\_Line("new\_base[" & Integer'Image(j) & "]=" & Integer'Image(arr(j)));

Ada.Text\_IO.Put\_Line("sigma=" & Long\_Float'Image(sigma));

Ada.Text\_IO.Put\_Line("--------------------J=" & Integer'Image(j) & "----------");

end loop;

Ada.Text\_IO.Put\_Line("---------------STEP 5---------------");

Ada.Text\_IO.Put\_Line("---------------STEP 6---------------");

--Set top(k) to top(k) -1

top(k) := top(k) - 1;

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(k) & "]=" & Integer'Image(top(k)));

move\_stack(base => base,

stack\_space => stack\_space,

top => top,

n => n,

arr => arr);

top(k) := top(k) + 1;

Ada.Text\_IO.Put\_Line("top[" & Integer'Image(k) & "]=" & Integer'Image(top(k)));

--Exception handling

temp\_boolean := insert\_stack(base => base,

top => top,

arr => arr,

m => m,

l0 => l0,

n => n,

k => k,

stack\_space => stack\_space,

val => val);

--Iterate over stacks

for j in Integer range 1..n loop

--set arr(j) to top(j)

arr(j) := top(j);

--Print Arr to the screen

Ada.Text\_IO.Put\_Line("Arr[" & Integer'Image(j) & "]=" & Integer'Image(arr(j)));

end loop;

Ada.Text\_IO.Put\_Line("---------------STEP 6---------------");

Ada.Text\_IO.Put\_Line("-------------REALLOC---------------");

return True;

end realloc;

end multistack;

**Main.adb**

with Ada.Text\_IO;

use Ada.Text\_IO;

with multistack;

--use multistack;

procedure Main is

--New data type(names)

type Names is (David,Joe,Bob,Devin,Elwin,Sagar,Ryan,

Kyle,Tyler,Minh,Adrian,Daniel,Jailene,

Jacob,Bryce,Jaylene,Ayran,Travis,Rahul,

Victor,Corey,Bipin,Frank,Nathan,Amber,Amy,

No,Hope,no\_name);

--New data type(months)

type MonthName is (January, February, March, April, May, June, July, August, September, October, November, December);

--record

type Date is record

Month : MonthName;

Day: Integer range 1..31;

Year: Integer range 1400..2020;

end record;

--Function to print the dates

function print\_date(d : in Date) return String is

begin

--Order of print\_dates Year/Month/Day

declare

temp : String := Integer'Image(d.Year) & " " & MonthName'Image(d.Month) & " " & Integer'Image(d.Day);

begin

--return print\_dates

return temp;

end;

end print\_date;

--generic package comparable to java class(names)

package names\_multistack is new multistack(item => Names,

image => Names'Image);

--generic package comparable to java class(months)

package date\_multistack is new multistack(item => Date,

image => print\_date);

-- Return substring of string from start\_index to end\_index

-- Arguments:

-- s which is the string to get substring from

-- start\_index which is the index where to start copy from

-- end\_index which is the index where to stop copying

-- Return:

-- substring

function split\_string(s : in String; start\_index: in Integer;end\_index : in Integer) return String is

temp : String(1..(end\_index - start\_index + 1));

counter : Integer := 1;

begin

for i in Integer range start\_index..end\_index loop

temp(counter) := s(i);

counter := counter + 1;

end loop;

return temp;

end split\_string;

function to\_name(s : in String) return Names is

begin

for i in Names'Range loop

if i = Names'Value(s) then

return i;

end if;

end loop;

return no\_name;

end to\_name;

--Gets date

-- Arguments:

-- Location which to extract date from

-- Return:

-- Date

function to\_date(s : in String) return Date is

d : Date;

start\_index : Integer := s'First;

end\_index : Integer := 0;

counter : Integer := 0;

k : Integer := 0;

begin

--iterate over string

for i in s'Range

loop

--if delimeter is found

if s(i)=' ' or counter=s'Length-1 then

--get end index

end\_index := i -1;

--special case for year

if k = 2 then

end\_index := i;

end if;

declare

temp : String := split\_string(s => s,

start\_index => start\_index,

end\_index => end\_index); --month or year or day as string

begin

if k = 0 then

d.Month := MonthName'Value(temp); --set month in date object

k := k + 1;

start\_index := i + 1;

elsif k=1 then

d.Day := Integer'Value(temp);-- set day in date object

k := k +1;

start\_index := i + 1;

else

d.Year := Integer'Value(temp);-- set year in date object

end if;

end;

end if;

counter := counter + 1;

end loop;

return d;

end to\_date;

--initialize variables

n : Integer;

m : Integer;

l0 : Integer;

input\_file : File\_Type;

lb : Integer;

ub : Integer;

begin

-- Print to screen

Ada.Text\_IO.Put\_Line("Please Enter Input File: ");

--get input

declare

input\_file\_name: String := Get\_Line;

--Get integer representation

begin

Open(File=>input\_file,Mode=>Ada.Text\_IO.In\_File,Name=>input\_file\_name);

m := Integer'Value(Get\_Line(input\_file));

n := Integer'Value(Get\_Line(input\_file));

l0 := Integer'Value(Get\_Line(input\_file));

lb := Integer'Value(Get\_Line(input\_file));

ub := Integer'Value(Get\_Line(input\_file));

declare

-- base : names\_multistack.Int\_Arr(1..n+1);

-- top : names\_multistack.Int\_Arr(1..n);

-- arr : names\_multistack.Int\_Arr(1..n+1);

-- tot\_mem : names\_multistack.item\_arr(lb..ub);

base : date\_multistack.Int\_Arr(1..n+1);

top : date\_multistack.Int\_Arr(1..n);

arr : date\_multistack.Int\_Arr(1..n+1);

tot\_mem : date\_multistack.item\_arr(lb..ub);

begin

-- names\_multistack.init\_stack(base => base,

-- top => top,

-- arr => arr,

-- size => n,

-- tot\_size => m,

-- l0 => l0);

date\_multistack.init\_stack(base=>base,

top=>top,

arr=>arr,

size=>n,

tot\_size=>m,

l0=>l0);

--Condition while not at last location

while not End\_Of\_File(input\_file) loop

declare

line : String := Get\_Line(input\_file);

stack\_num : Integer := Integer'Value(split\_string(line,2,2));

begin

--Condition for if deletion in multistack

if line(1) = 'D' then

date\_multistack.delete\_stack(base=>base,

top=>top,

k=>stack\_num,

stack\_space=>tot\_mem);

else

declare

v : String := split\_string(line,4,line'Last);

val : Date := To\_Date(v);

begin

--Condition for if insertion in multistack

if date\_multistack.insert\_stack(base=>base,

top=>top,

arr=>arr,

m=>m,

l0=>l0,

n=>n,

k=>stack\_num,

stack\_space=>tot\_mem,

val=>val) = False then

exit;

end if;

end;

end if;

end;

end loop;

-- while not End\_Of\_File(input\_file) loop

-- declare

-- line : String := Get\_Line(input\_file);

-- stack\_num : Integer := Integer'Value(split\_string(line,2,2));

-- --v : String := split\_string(line,3,line'Last);

-- --val : Names := To\_Name(v);

-- begin

-- if line(1) = 'D' then

-- names\_multistack.delete\_stack(base => base,

-- top => top,

-- k => stack\_num,

-- stack\_space => tot\_mem);

-- else

-- declare

-- v : String := split\_string(line,3,line'Last);

-- val : Names := To\_Name(v);

-- begin

-- if names\_multistack.insert\_stack(base => base,

-- top => top,

-- arr => arr,

-- m => m,

-- l0 => l0,

-- n => n,

-- k => stack\_num,

-- stack\_space => tot\_mem,

-- val => val) = False then

-- exit;

-- end if;

-- end;

-- end if;

-- end;

--

-- end loop;

end;

--close file

close(input\_file);

end;

end Main;