COSC 3319 Data Structures
Lab 1 Spring 2016 Burris

Due: Thursday February 18th. I will accept this lab without taking off for being late through Thursday February 25th. THIS LAB MAY NOT BE SUBMITTED FOR CREDIT AFTER February 25th! Expect additional labs to be assigned during this time frame. You may use Windows, UNIX (any flavor including Apple) or Chrome operating system. Other operating systems will be considered on request.

This lab is meant to push you beyond your current level of expertise to new planes of achievement, i.e., professional grade. It purposely contains elements that will require you to research new solution techniques. A goal of the assignment is to force you learn to research and master new technology either on your own or with a minimum of external help. I am always available for questions. Labs must be submitted in an envelope sufficiently large to hold your results without having to fold the lab materials. Write your name and "COSC 3319" in the upper left hand corner of the envelope opposite the side the flap is folded. You may use the same envelope for all labs during the semester.

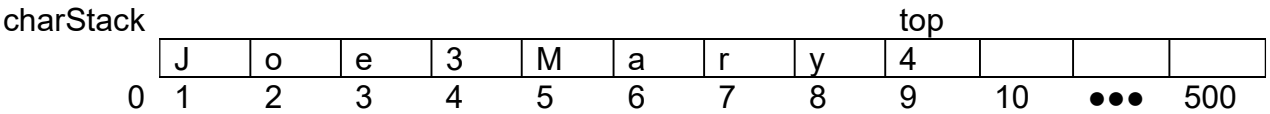
You must clearly state on the first page of your program which grading option you have successfully completed. Programs failing to state the grading option will be graded as "C" option programs. Submit a printed copy of all code, any data files, and results. Do not embarrass yourself or me with screen shots. This lab is not technically challenging with respect to data structures. Its primary purpose is to ensure you gain the required background to complete "A" option work on future labs. Step up to "professional grade!"

Additional labs will be assigned prior to the due date. Do not procrastinate finishing this first assignment. It will help you learn the new required language technology to implement the advanced data structures labs. This lab is designed so you can start by implementing the "D" option first. The "D" option can be converted to the "C" option, then the "B" option, and finally the "A" option with minimal loss of effort. In essence, you tell me what you are worth as a student. Be "Professional Grade!" Do the "A" option.

Most of the grading options give you a choice of languages. You are encouraged to use Ada for several reasons. First, if you ever work with a company contracting with DOD (Department of Defense) or European countries, knowledge of Ada is an advantage. You should be able to use Ada, Java, Groovy, C++ or C# for the "B" and "A" options on all labs. You may use Ada, C++, C#, PHP, Java, assembly or COBOL for the "C" option on all labs. **Due to experience from previous semesters, you may not use Python**. This class is oriented towards a real time environment with some latitude. Python does not support the abstractions required for the advanced requirements on most labs. Experience indicates Python programmers spend excessive time researching Python to overcome its inherent short falls with respect to meeting real time restrictions. COBOL may be used for the "B" and "A" options if you have access to an object oriented COBOL compiler. Substantial examples using Ada will be utilized in class over the next few weeks.

If you work for bleeding edge technology companies, you will have to learn to change programming languages like you change your clothes. There are two required courses in Java (COSC1436 and COSC1437) and it is used in several other courses. "C/C++" is taught in COSC 2347 and used in several other courses. Ada will be used in this class, Object Oriented technology, and Operating Systems when taught by me. The more languages you know prior to graduation, the easier it will be to learn new languages on the job after graduation. Most companies will expect you to learn new languages on your on time using your own resources.

PROBLEM SPECIFICATION: We have been asked to write a program for scientist providing support for a research radio telescope. The telescope receives mostly static but occasionally receives strings of characters which must be saved for study at a later time. Our customer wishes to determine if the character strings are random accidents or an attempt at communication by an extraterrestrial life form. The customer desires to store the characters in a stack with the number of characters in the string as the top item in the stack for each string. We may assume that no string will exceed 255 characters in length. In fact the strings average 5.6 characters in length with the longest string detected to date consisting of 122 characters. For example, the strings "Joe" and "Mary" would appear as follows in a stack of 500 characters (M = 500) with the top of stack currently at location 9.



Each time the customer invokes "push" they will either place one character of the string in the stack or a character/integer representing the number of characters in the string. The count will be inserted into the stack after processing all alphabetic characters. The stack is considered empty when top = 0.

The first time "pop" is called it will return a character/integer indicating the number of characters in the string. Pop will then be called successively to remove the character string from the stack (returned in reverse order). Upon overflow, an error message should be generated and the program terminated. Underflow should be ignored and program operation continued.

The following algorithms have been suggested for stack operations.

<table>
<tr><td>Insert in stack::
If top < M then
top ← top + 1
charStack(top) <- y
Else
Report overflow
End If</td><td rowspan="2">Your implementation <u>“must”</u> return an indication to the user if the operation succeeded or failed! At least one of the procedures or functions should use a Boolean for this purpose!</td></tr>
<tr><td>Delete from stack::
1) If top = 0 then
report underflow.
process next operation from radio telescope.
End If;

2) Y ← character/integer on top of stack.
Process character in Y (from pop).</td></tr>
</table>

The main procedure should continue to process transactions until all transactions have been processed or the program must be terminated due to overflow.

Hint 1: Assume the name of your program is Pgm1.exe. If you type "Pgm1" at the DOS prompt, input is normally expected from the keyboard and results are printed on the terminal. The command "Pgm1 > file1" would expect all input to come from the keyboard but the results would be routed to a disk file in the current directory named "file1." The command "Pgm1 < raw1 > results" would obtain input (as stream IO) from the file "raw1" and place the output in the text file "results" in the current directory.

Hint 2:
Normally Ada will not allow an integer to be assigned to a character variable as bits must be truncated. Similarly, Ada will not normally allow a character to be assigned to an integer as there are not enough bits (padding must occur). In both cases, Ada will flag a probable error at compile time letting the programmer know they have probably made a logic error. To tell the compiler you really mean to perform the indicated operation, the generic package “Unchecked_Conversion” should be instantiated to allow the conversion as shown below. There is no actual runtime penalty for most instantiations of Unchecked_Conversion (no run time function call) resulting in CPU overhead when using a good Ada compiler. The generic conversion is simply allowed at compile time as in the example below.

```
with Ada.Text_IO; use Ada.Text_IO;  -- read and write characters.
with Unchecked_Conversion;  -- standard package with every validated Ada translator

procedure ConvertCharacterInteger is
   -- To read and write 16 bit integers on the PC compiler.
   package MyInt_IO is new Ada.Text_IO.Integer_IO(integer);
   use MyInt_IO;
```

```
   -- instantiations to convert between integer and character formats, 16 bits versus 8 bits.
   function integerToCharacter is new Unchecked_Conversion(Integer, Character);
   function characterToInteger is new Unchecked_Conversion(Character, Integer);

   c1, c2: Character;
   int1, int2: Integer;

   begin
     c1 := 'A';
     -- int1 := c1;  --error in Ada, strongly typed, suspects programmer error.
     int1 := characterToInteger( c1 );  -- Signal compiler to allow conversion.
     put(" int1 = "); put(int1,4); put("  ");
     put("c1 = "); put(c1); new_line(2);

     int2 := 66;
     -- c2 := int2;  -- error
     c2 := integerToCharacter(int2);
     put(" int2 = "); put(int2,4); put("  ");
     put("c2 = "); put(c2); new_line(2);


end ConvertCharacterInteger;
```

In application programming, it is occasionally desirable to treat the same unit of memory at different times as a different data type!

Frequently used conversion trick in assembly, “C” and other languages.

ASCII Conversions

Character	Decimal	Integer	Integer (32 bits) Binary
‘0’	48	0	0-0000
‘1’	49	1	0-0001
‘2’	50	2	0-0001
‘3’	51	3	0-0011
‘4’	52	4	0-0100
‘5’	53	5	0-0101
‘6’	54	6	0-0110
‘7’	55	7	0-0111
‘8’	56	8	0-1000
‘9’	57	9	0-1001

Assume we wish to convert a 32 bit one digit integer to an 8 bit ASCII character. This may be accomplished by adding the character ‘0’ (48 decimal) to the integer and dropping the leading 24 zeros. As an example the 32 bit integer 3 may be converted to a character by adding the character ‘0’ (or decimal 48) then truncating the leading 24 bits.

Ex: 3 + 48 = 3 + ‘0’ = 51 => ’3’ in ASCII

Alternately an ASCII digit represented as character may be converted to an integer by subtracting the character '0' (or 48) and padding the 24 bits to the left with zeros.

Ex: '3' – 48 = 51 – 48 = 3 => 3 in base 10 or 11 in binary.

Using "C:"
ch: char = '3';
int1: int = 0;

int1 = (int)ch - 48; //first coerce/cast the character to an integer then convert to integer 3.
Or alternately:
int1 = int(ch) – 48; //function form of casting.

ch = char(3 + 48); // yields the character 3.
Or
ch = (char) (3 + 48); // yields the character 3.

In COBOL, use the "redefines verb."

"D" Option (best grade is 60).

You may use Ada, C++, C#, Perl, Java, PHP, COBOL, Groovy or any other programming language translator publicly supported by SHSU Computer Services to implement the "D" option. You are encouraged to use functions / procedures properly but may write the program as monolithic code. Process all of the following transactions. **Prompt the user for the size of the character stack (M) at runtime and dynamically allocate space for the user stack in the system stack** using the technique shown in Ada, C++, or Java. *You specifically may not use "new, malloc," or any other operator, which allocates space in the heap in any language. Clearly mark this section of your code with a highlighter!* Print each transaction as it is processed. I recommend the use of a data file. It will save you time and effort.

Data Set 1:

Stack size for dynamic allocation is 20 characters.

Insert Joe (Enter Joe at the keyboard followed by a delimiter such as a space or # one character at a time, all at once, or in any reasonable manner of your choosing. Your main program [radio telescope] should count the number of characters as they are placed in the stack. Once the string is placed in the stack, the number of characters in the string must be placed in the stack).

Insert moT

Insert Betty

Pop and print the top string in the stack (output will be 5 followed by ytteB).

Pop and print the top string in the stack (output will be 3 followed by Tom).

Pop and print the top string in the stack.

Pop and print the top string in the stack

Insert Larry

Insert Sarah

Insert Bob

Pop and print the top string in the stack.

Insert Harold

Insert Sue

Submission: Submit a cover page with your name indicating the "D" grading option. The cover page should be followed by the "D" option results then the "D" option code.

"C" Option (best grade is 70):

You may use Ada, C++, C#, PHP, Groovy, Ruby or Java to implement the "C" option. I will even accept Python for the "C" option so long as you explicitly code all requirements as opposed to taking advantage of language facilities. You need not do the "D" option. Rather implement the "D" option algorithms as procedures or functions encapsulated in a package / class. The "push" and "pop" operations must return an indication overflow or underflow has occurred and your program must detect and respond in an appropriate fashion to the condition. You are encouraged to consider notifying the main program overflow or underflow has occurred using a Boolean variable. Again, **you should prompt the user for the value of M at runtime and allocate the space for the stack in the *stack* at runtime**. Process the "D" option data set. If you use COBOL, push and pop must be implemented as subroutines with appropriate parameter passing.

Submission: Submit a cover page with your name indicating the "C" grading option. The cover page should be followed by the "D" option results then the "C" option code (code for telescope followed by package / class specifications then package / class bodies).

"B" Option (best grade is 80):

You must use Ada, C++, Java, Groovy, Ruby or PHP to implement the "B" option. You are encouraged to use Ada, it requires by far the least programming effort. You need not do the "D" or "C" options. Rather implement the "D" option algorithms using a generic package that allows the user to specify the data type to be saved in the stack and desired size of the stack. The stack should default to 5 units when the size is not specified by the user. Write an appropriate error message if the stack is full (do not store the item causing overflow) or empty then continue processing data. Do not terminate. You may use examples in the Hymnal as a guide in writing this code.

First: Process the "D" option data set and submit the results.
Second: Instantiate a stack to store integers and process the following data set. Integers should be stacked till the sentential "0" is encountered. Upon encountering the sentential, the number of integers should be placed in the stack.

Data set 2:
Stack size for dynamic allocation is 15.
Insert 7, 18, 10, 25, 3, 46, 0. (After the operation, the stack should contain 7, 18,10, 25, 3, 46, and 6 as the top element).
Pop and print the top string of integers in the stack. (Should print 6, 46, 3, 25, 10, 18 and 7)
Insert 32, 12, 456, 7, 9, 0.
Insert 5, 17, 0.
Insert 3, 0.
Insert 42, 23, 67, 85, 0.
Pop and print the top string of integers in the stack (4, 85, 67, 23, 42).
Pop and print the top string of integers in the stack. (1, 3)
Insert 34, 22, 56, 78, 54, 6, 17, 67, 86, 0.

Submission: Submit a cover page with your name indicating the "B" grading option. The cover page should be followed by the "D" option results followed by the "B" option data set results. The "B" option code should appear next (code for telescope) followed by package / class specifications then package / class bodies.

"A" Option (best grade is 90):

Ada encourages programmers to define and use their own data types. C++ and Java restrict programmers to intrinsic data types in convenient generic (template) instantiations. Ada allows convenient instantiation with any programmer data type. You must use Ada, C++, C# or Java for the "A" option.

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

```
-- have compiler write I/O routines to print programmer defined enumeration type.
package MonthNameIO is new Ada.Text_IO.Enumeration_IO(MonthName);
use MonthNameIO;
-- I/O routines to read and write integers.
Package IntegerIO is new Ada.Text_IO.Enumeration_IO(Integer);
Use IntegerIO;

type DateType is record
   month: MonthName;  day: integer range 1..31;  year: integer;
end record;

aDate: DateType := (January, 15, 1947); -- A typical declaration and assignment.
```

First process the "C" option data set followed by the "B" option data set. Finally process the following data set.

Data set 3:
Stack size 8
Insert January 15, 1952; February 23, 1492, September 1, 2004 (you are free to select a method to terminate the string of dates).
Insert October 12, 2003 (one date string of dates).
Insert February 4, 1777, December 24, 2003, June 12, 2004.
Pop and print string of dates neatly.
Pop and print string of dates neatly.
Pop and print string of dates neatly.
Pop and print string of dates neatly.
Insert September 17, 1623, August 11, 2002.
Pop and print string of dates

Submission: Submit a cover page with your name indicating the "A" grading option. The cover page should be followed by the "D" option results followed by the "B" option data set results then the "A" option results. The "A" option code should appear next (code for telescope) followed by package / class specifications then package / class bodies.

A skeleton procedure to print a date follows:

```
procedure printDate( adate: in DateType) is
begin
        put(adate.month);  put(adate.day,3); put(" "); put(adate.year); new_line;
end printDate;
```

I recommend the most recent edition of "Programming in ADA" by John Barnes, for example ISBN 0-201-34293-6 for the second edition. All editions of Branes text starting with the 2nd edition are sufficient for the course. Almost any Ada book will work (Ada 2012, Ada 2010, or Ada 95). I recommend "Java How to Program" (6th edition or latter) by Deitel and Deitel for Java, ISBN 0-13-148398-6. "C/C++" text covering templates and inheritance will be adequate for the material covered in labs. Select a "C/C++" text oriented to the platform on which you plan to code, i.e., Windows, Linux, Chrome, etcetera.

Please remember use of work by other students from this semester or previous semesters will generally result in an "F" for the course! You are however encouraged to discuss solutions with each other and provided reasonable help. One of the best ways to learn is from your fellow students or helping fellow students. You may not however work together as a team to solve lab problems.

Hint: Basic structure for main program:

```
Loop
        Get(stackType);
        Get(stackSize);
        Case(stackType) is
                characterType:
                        declare  -- dynamic allocation of space in stack
                                generic instantiation of stack using stackType and stackSize
                        begin
                                operations on stack
                        end  -- deallocation of data structure from stack

                integerType:
                        declare  -- dynamic allocation of space in stack
                                generic instantiation of stack using stackType and stackSize
                        begin
                                operations on stack
                        end  -- deallocation of data structure from stack

                dateType:
                        declare  -- dynamic allocation of space in stack
                                generic instantiation of stack using stackType and stackSize
                        begin
                                operations on stack
                        end  -- deallocation of data structure from stack

        end case;
End Loop;
```