Due July 20 at midnight to eCampus

First Name Carson Last Name Hanel UIN 826003149

User Name Chingy1510 E-mail address Chingy1510@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)			
Printed material			
Other Sources	Lecture Material		

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name Carson Hanel Date 7/20/2017

Programming Assignment 2 (140 points)

1. (30 pts) Implement a stack ADT using STL vector to support the operations: `push()`, `pop()`, `top()`, `empty()`.
2. (10 pts) Test the operations of the stack class for correctness. What is the running time of each operation? Express it using the Big-O asymptotic notation.
   (a) The Push and Pop operations are both O(1) time. Because the stack is not meant to be resized, you don't have to worry about amortization.
3. Stack Application

   Use the implemented stack class as an auxiliary data structure to compute spans used in the financial analysis, e.g. to get the number of consecutive days when stack was growing.

Definition of the span:

Given a vector X, the span $S[i]$ of $X[i]$ is the maximum number of consecutive elements $X[j]$ immediately preceding $X[i]$ such that $X[j] \leq X[i]$

Spans have applications to financial analysis, e.g., stock at 52-week high

1. Example of computing the spans S of X.

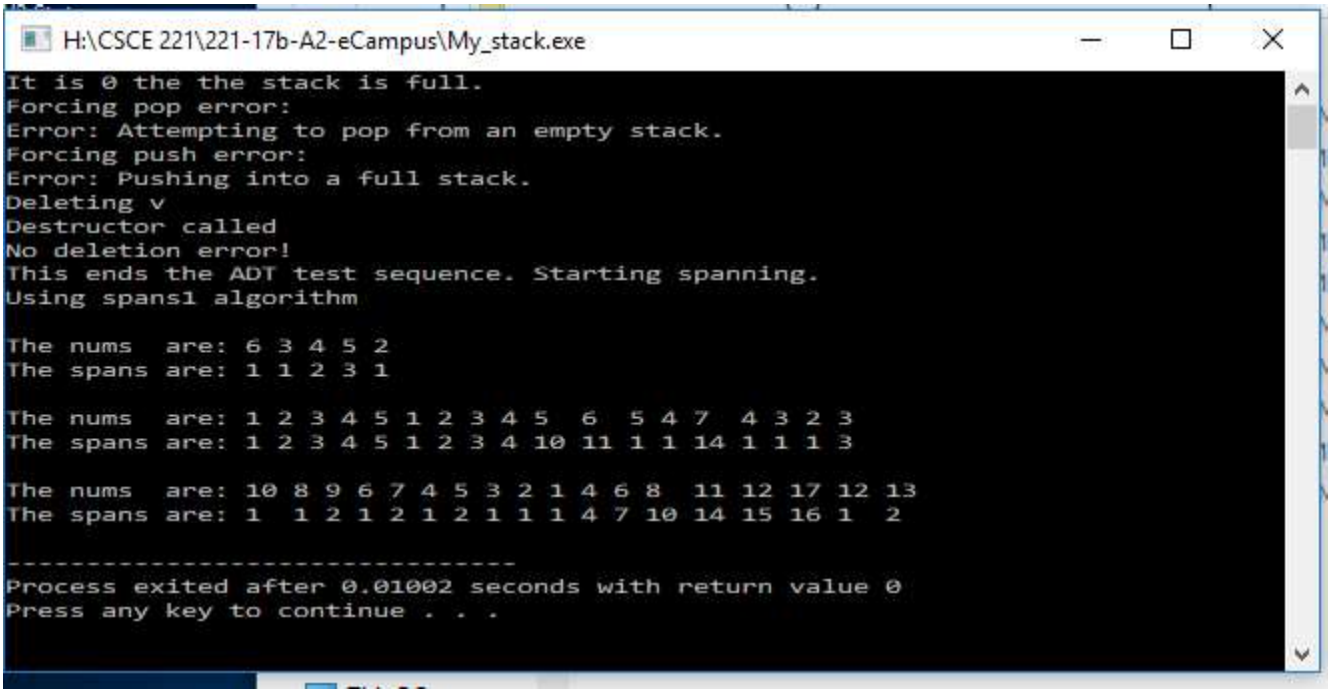
X	6	3	4	5	2
S	1	1	2	3	1

2. (10 pts) Compute the spans based on the definition above:

```
Algorithm spans1(x)
Input: vector x of n integers
Output: vector s of spans of the vector x
for i = 0 to n-1 do
   j = 1
   while (j <= i ∧ x[i-j] <= x[i])
      j =  j + 1
   s[i] = j
return s
```

1. (10 pts) Test the algorithm above for correctness using at least three different input vectors.

(a)



2. (10 pts) What is the running time function of the algorithm above? The function should define the relation between the input size and the number of comparisons perform on elements of the vector. How can you classify the algorithms using the Big-O asymptotic notation and why?

(a) The running time function of the algorithm above is at worst O(n^2). Every iteration of the for-loop, there are, at most, 2(n - 1 - i) comparisons done by the while loop, and at LEAST one comparison. Giving this function the running time of f(n) = n * 2(n - 1)/2 or O(n^2)

(b) You can classify algorithms using Big-O asymptotic notation because the Big-O is a representative set of all subset functions of equal power modified by some constant.

3. (20 pts) Compute the spans using a stack as an auxiliary data structure storing (some) indexes of x.

   Algorithm spans2:

   (a) We scan the vector x from left to right

   (b) Let i be the current index

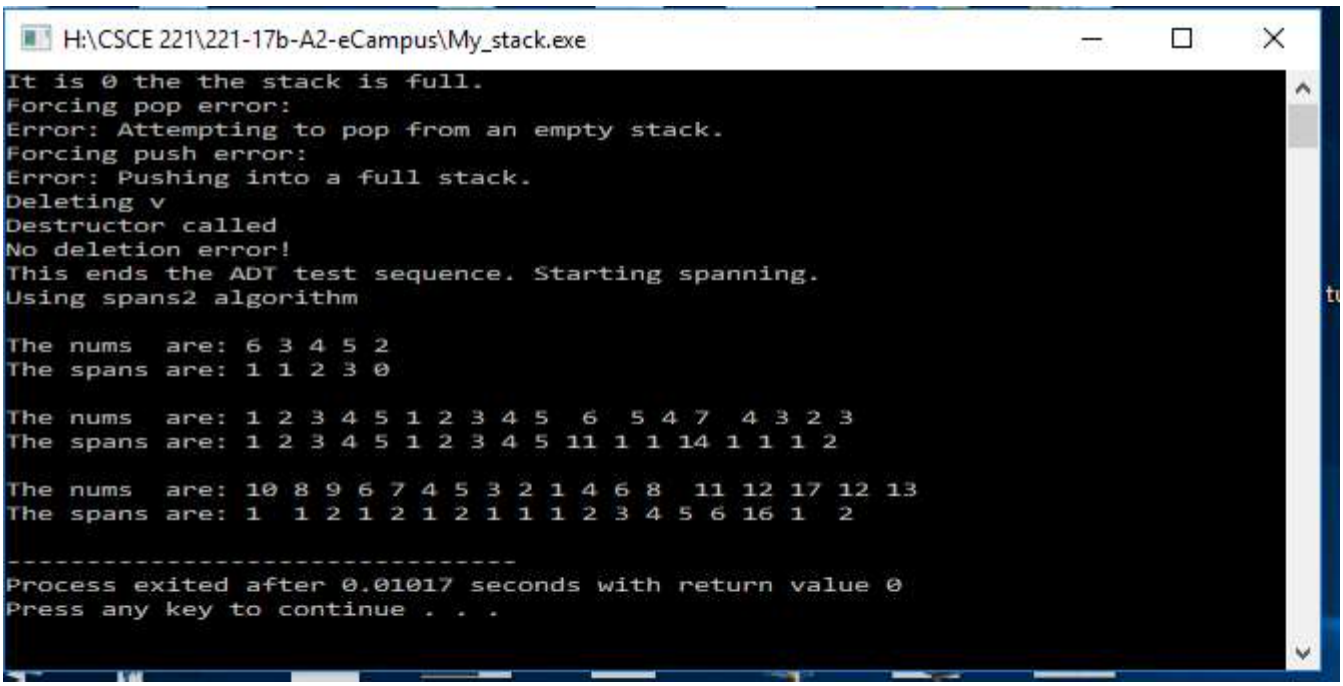
   (c) Pop indices from the stack until we find index j such that x[i] $<$ x[j] is true

   (d) Set s[i] = i $-$ j

   (e) Push i onto the stack

4. (10 pts) Test the second algorithm (spans2) for correctness using at least three different input vectors. Your program should run correctly on TA's input.

   (a)



5. (10 pts) What is the running time function of the second algorithm? The function should define the relation between the input size and the number of comparisons perform on elements of the vector. How can you classify the algorithms using the Big-O asymptotic notation and why? Compare the performance of both the algorithms.

   (a) For sake of ease, the runtime function of the pseudocode algorithm, it would be that there is a perfect f(n) = n-1 = O(n).

   (b) You can classify algorithms using Big-O asymptotic notation because the Big-O is a representative set of all subset functions of equal power modified by some constant.

   (c) Comparison:

      i. The first algorithm is inefficient. Given that the maximal peak within the set is the right-most member of the input vector, it would take n-1 comparisons just to calculate for the spans.

      ii. The second algorithm, however, takes advantage of the stack data structure which keeps track of the lefternmost known maximum peak, as well as all subsequent sub-maximal peaks.

         A. This is important, because all peak locations in a dataset require a constant time operation of 1 push and 1 pop AT MOST.

B. Instead of, like spans (vs. spans2), iterating through the dataset calculating spans arbitrarily for each data member,

C. we are introducing assumption into the programming process. We know that, given a lefternmost maximal peak and all internal sub-maximal peaks,

D. our newly found peak has to fit in SOMEWHERE. As you iterate backwards through peaks, instead of iterating by individual data members, you're

E. iteration backwards entire spans. So, the point. We know that, if a newly found peak is greater than some other peak, between itself and its neighboring

F. peak, you can know with 100% certainty that all datamembers to the left of that peak and right of the peak to the left of it (long winded) are less than your current peak.

G. Why is this important? Well, now all the sudden instead of iterating through the entire vector, you're simply checking which peaks your new peak is greater than.

H. It's really intuitive, and I had a ton of fun figuring it out.

6. (10 pts) Programming style, and program organization and design: naming, indentation, whitespace, comments, declaration, variables and constants, expressions and operators, line length, error handling and reporting, files organization. Please refer to the PPP-style document.

7. Instructions

   (a) Your files should be arranged as below

      i. Declaration of `My_stack` class in `My_stack.h`
      ii. Definition (implementation) of `My_stack` class in `My_stack.cpp`
      iii. Algorithm implementations and the `main()` function in the file `Application.cpp`

   (b) Compile your program by

   ```
   g++ -std=c++11 *.cpp
   ```

   or

   ```
   make all
   ```

   (c) Run your program by executing

   ```
   ./Main
   ```

   (d) Testing code in `Application.cpp` and collect output data for your report.

8. Submission

   (a) Tar the files above and any extra files. Please create your tar file following the instructions.

   (b) "turnin" your tar file to eCampus no later than **July 20**.

   (c) (20 points) Submit a hard copy of cover page, report (see below), `My_stack.h`, `My_stack.cpp` and `Application.cpp` in lab to your lab TA. Find a cover page .

      i. Typed report made preferably using "LyX/LaTeX"

         A. (2 pts) Program description; purpose of the assignment

         B. Program description: The program is a comparison study between two span-finding algorithms and their complexities.

         C. Purpose: The purpose of the assignment was learning to implement an STL Stack, and use it in a helpful application. At the same time, it was a good assignment

         D. to really get a look at more efficient ways of doing things. I enjoy watching lectures comparing algorithms from decades past and the most optimal solutions of

         E. the scientists before us. It makes me wonder what kind of crazy efficient solutions we'll come up with in the future.

         F. (4 pts) Data structures description

            - Theoretical definition
              - The stack, theoretically, is a first in, last out functioning data structure that is useful for holding and accessing data members in a particular ordered way.
            - Real implementation
              - To implement the stack, you have to have a few key parts:
                * Private members: Things your stack needs to know are where the top element is, what the stack's capacity is, and where to look for the data (a dynamic array).
                * Public members: Ways you need to be able to interact with your stack include functions:
                  · top(): Returns the top member of the stack, but doesn't pop it off.
                  · In my implementation, I used peek(), but that's because I was holding the location of the stack's top member with variable name top.
                  · push(): IF the stack is not full, pushes the object into the first open slot and increased the top variable by 1.
                  · pop(): IF the stack is not empty, returns the top element's value and decreases top by 1.
                  · empty(): IF top is -1, return true, that yes, the stack is empty.
                  · Pretty much, those public and private members make up the STL Stack used in the assignment.
              - To implement a spans algorithm you need (if this is what you're asking?):
                * a linear input of some sort
                * an iterative method capable of allowing comparisons of some sort
                  · for loop, while loop, recursive statement

    - either a method of comparing the distance between the iterator and some object of input, or that coupled with some form of memory of objects.
      - using j as a comparison iterator between data member spans, or
      - doing that, coupled with a stack, queue, some linear form of recalling previously seen peaks.
    - That's pretty much the structure. Oh, and a new object of some sort (vector) keeping track of each data member's spans.
- Analysis of the best and worst scenarios for computing spans.
  - For spans1:
    - Best case would be n-1 operations to set all spans. (see below explaination)
    - Worst case would be n*(n-1)/2 = O(n^2).
  - For spans2:
    - The best case would be a constantly decreasing dataset, wherein n-1 comparisons are used, and all datamembers are pushed into the stack as sub-maximal peaks.
    - The worst case would be the best case, followed by a datamember higher than the 0th value of the array, causing you to have to compare and pop out all n-1 sub-maximal peaks.

ii. (2 pts) Instructions to compile and run your program; input and output specifications

A. Compilation instructions:

B. My_stack.h is the implementation of the algorithm and the STL Stack

C. My_stack.cpp is the testing driver for My_stack.h

D. Put them in a folder together

E. g++ -std=c++11 My_stack.h My_stack.cpp

F. ./a.out

G. Watch the magic (if it zips through it just throw a system("pause"); in there)

iii. (2 pts) Logical exceptions (and bug descriptions)

A. Push: attempting to push into a full stack throws an exception of: "Pushing into a full stack."

B. Pop: attempting to pop from an empty stack throws an exception of: "Attempting to pop from an empty stack"

C. Top(peek): attempting to check the top value from a stack that is empty throws an exception of: "Attempting to check top from an empty stack"

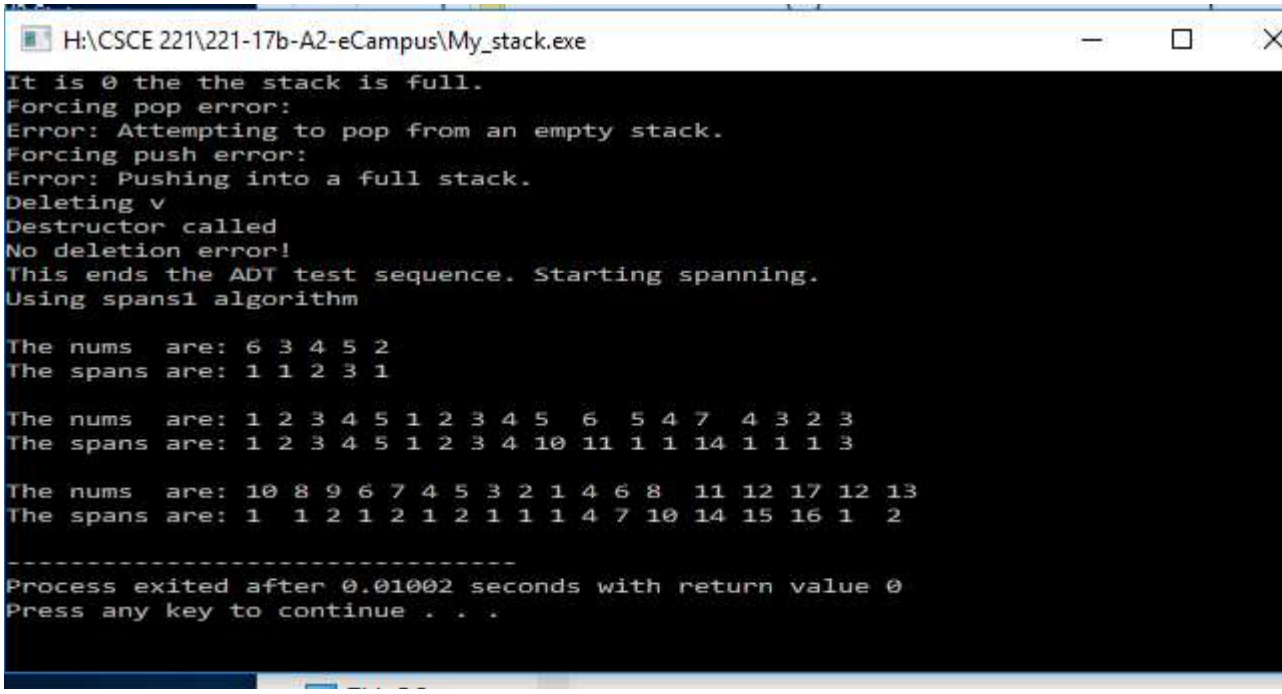
iv. (5 pts) C++ object oriented or generic programming features, C++11 features

A. Using STL Stack, pointers, vectors, cout, heap access of object member functions, etc.

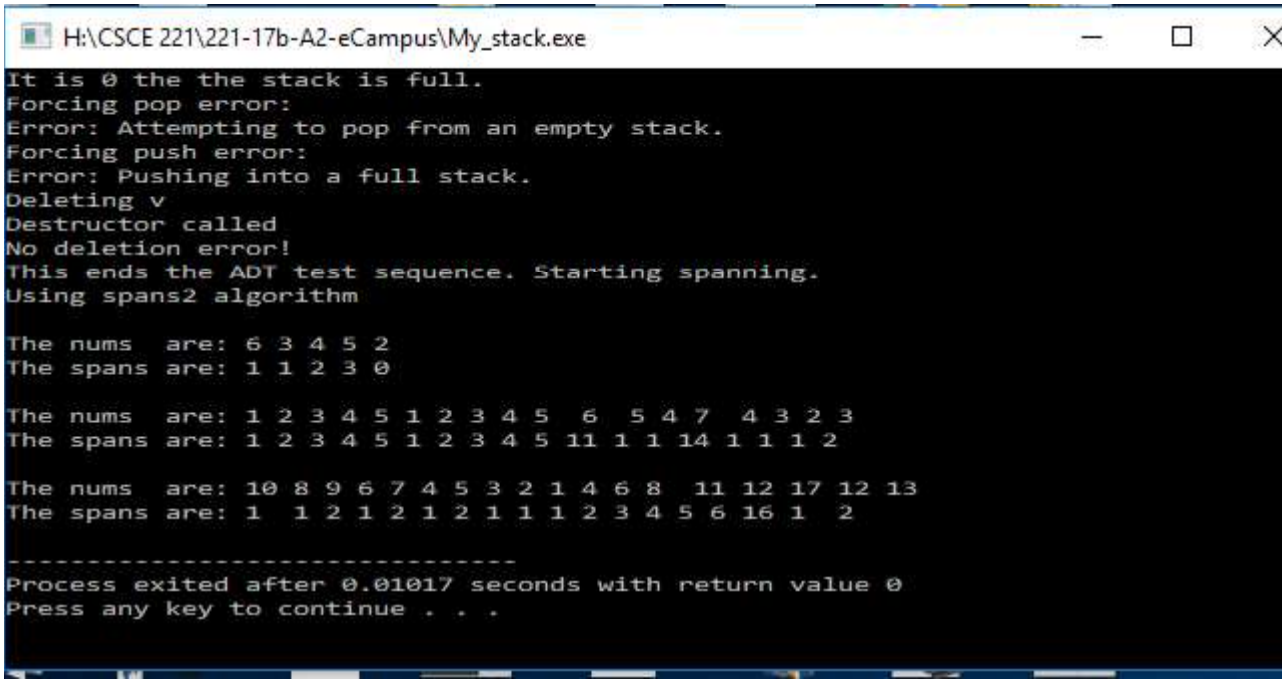
v. (5 pts) Testing results

A. Testing Results:

B.


H:\CSCE 221\221-17b-A2-eCampus\My_stack.exe
It is 0 the the stack is full.
Forcing pop error:
Error: Attempting to pop from an empty stack.
Forcing push error:
Error: Pushing into a full stack.
Deleting v
Destructor called
No deletion error!
This ends the ADT test sequence. Starting spanning.
Using spans1 algorithm
The nums are: 6 3 4 5 2
The spans are: 1 1 2 3 1
The nums are: 1 2 3 4 5 1 2 3 4 5 6 5 4 7 4 3 2 3
The spans are: 1 2 3 4 5 1 2 3 4 10 11 1 1 14 1 1 1 3
The nums are: 10 8 9 6 7 4 5 3 2 1 4 6 8 11 12 17 12 13
The spans are: 1 1 2 1 2 1 2 1 1 1 4 7 10 14 15 16 1 2
Process exited after 0.01002 seconds with return value 0
Press any key to continue . . .

C.


H:\CSCE 221\221-17b-A2-eCampus\My_stack.exe
It is 0 the the stack is full.
Forcing pop error:
Error: Attempting to pop from an empty stack.
Forcing push error:
Error: Pushing into a full stack.
Deleting v
Destructor called
No deletion error!
This ends the ADT test sequence. Starting spanning.
Using spans2 algorithm
The nums are: 6 3 4 5 2
The spans are: 1 1 2 3 0
The nums are: 1 2 3 4 5 1 2 3 4 5 6 5 4 7 4 3 2 3
The spans are: 1 2 3 4 5 1 2 3 4 5 11 1 1 14 1 1 1 2
The nums are: 10 8 9 6 7 4 5 3 2 1 4 6 8 11 12 17 12 13
The spans are: 1 1 2 1 2 1 2 1 1 1 2 3 4 5 6 16 1 2
Process exited after 0.01017 seconds with return value 0
Press any key to continue . . .