CIS-227 Data Structures and Algorithms

Review: ADT – Stack

1.
   a. Define the ADT: stack
   b. Explain what is contained in a stack frame
   c. Name three applications of stacks

2. Complete the following (answer: true or false):

   a. A stack is used by the system when a function call is made
   b. A stack can become full during program execution
   c. The postfix expression: 3 4 5 * + evaluates to: 23
   d. The postfix expression: 5 2 * 6 3 * + evaluates to:
      tops(s,x), is equivalent to: push(s,x); pop(s,x);

3. Show how an opera do stack is formed and ALL of its various stages computing the value of the postfix expression:
   a. 1 2 3 + * 3 2 1 - + *
   b. 8 5 2 + - 3 8 1 + + * 2 + 2 +
   c. 1 3 2 - + 2 6 3 / * + 3 + 2 +
   d. 4 1 2 * + 5 1 4 + * -

4. Transform the infix expression to postfix form. Do NOT use the infix to postfix conversion algorithm. (Manual).

   a. (A + B) * (C – D) + E * F
   b. (V + A) * (C * (A – T) + I) – O * N
   c. E * (A + B) – D / (G – F)
   d. A + (((B – C) * (D – E) + F) / G) * (H – J)
   e. C * (O + M) – P * I + L – (E + R)

5. Use the infix to postfix conversion algorithm to transform the infix expression to postfix form (use stacks):
   a. A * B + C * D – E
   b. A + B * C – D * E
   c. G * (B + C) – (D + E)
   d. H * (I – J) + K * (X + Y)
   e. A – (B + C) + D * (E + F)

6. Transform the postfix expression to infix form. Show ALL the steps.
   a. A B * C D E / - +
   b. A B - C + D E F - + *
   c. A B C D E - + / * E F * -

7. Given that a stack already exists and given the stack toolkit:
   push(s, i) pop(s, i) top(s, i)

   Write just a sequence of statements to assign to : answer a copy of the third element in the stack (the top element is the first one) leaving the stack unchanged (assume the has at least three items). Do not write a function and do not declare any variables.

8. Given that s1 and s2 are stacks of integers, determine the output that results when the following code is executes:

```
create_stack(s1);
create_stack(s2);
for( ct = 1; ct <= 10; ++ ct )
        push(s1, ct);
while ( !empty(s1) ) {
        pop(s1, ct);
        if ( ct % 2 != 0 )
                push(s2, ct);
}

while ( !empty(s2) ) {
        pop(s2, ct);
        cout << ct << "   ";
}
cout << endl;
```

9. Briefly explain why a stack is formed when a call is made to a function during program execution. What information must be saved?

10. Briefly explain the advantages of writing "intermediate level functions" (toolkit functions) like push, pop, etc. and then using them to write applications programs.

11. Write the definition code for a function that passes in a stack and returns (using a return statement) the number of items in the stack (the stack size).

    a. Assume that this function is to be a toolkit function in the implementation of the ADT stack.
    b. Assume that this function is not a toolkit function.

12. Given the declaration:

```
struct STACK {
        INFO_RC     i;
        int         top;
};
```

Write the definition source code for the following functions in the implementation of the ADT stack:

a.	create_stack(s)	d.	tops(s, i)
b.	empty(s)	e.	push(s, i)
c.	pop(s, i)	f.	purge(s)

13. Indicate the Big-0 "run-time" of each of the following:

    a. Deletion from a stack that is implemented using an array with the top of the stack always maintained in the component 0. Thus items are always inserted and/or deleted at component 0).
    b. Deletion from a stack that is implemented using an ordinary array with the top of the stack always maintained in the first available component of the array (the stack grows "upward" starting at component 0).
    c. Print out all of the elements in a stack.
    d. Determine the number of items currently in the stack. (stack size is not a toolkit function).

14. Given:

```
struct INFO_RC {
        int     id;
        char    name[30];
        float   income;
);
```

and the toolkit functions:

```
create)stack(s)'     top(s, i);
pop(s, i)            empty(s);
push(s, i);
```

Use some or all of them to write the definition code for the following functions (note these are NOT toolkit functions and that they should NOT directly access the data structure that represented the stack).

a. Passes in a stack and interchanges the top two items

b. Passes in a stack and returns (as an argument) the second item in the stack. It does not remove the item from the stack. Include a Boolean flag for the case of too few items.

c. Passes in a stack and returns (as an argument) the bottom item in the stack. It does not remove the item form the stack. Include a Boolean flag for the case of an empty stack.

d. Passes in a stack and returns (as an argument0 the mean income of all of the items in the stack.

e. Passes in a stack and returns (as an argument) the maximum income of all of the items in the stack.

f. Passes I a stack and a positive integer n; returns (as an argument) the nth item in the stack. It does not remove the items from the stack. Include a flag for the case of too few items.

g. Passes in a stack and returns another stack that is the first stack inverted (the top is now the bottom).