CS271: Data Structures
Stack Worksheet

1. Design and implement a stack class for integers using an array to hold the data. Use a default size of 100 items for the array, implement overflow and underflow exceptions for improperly accessing a full or empty stack. Include methods to push a new item on the stack and to pop (remove) and item from the stack. Add methods to check if the stack is empty or full.

2. Add the ability to specify the stack size as a parameter for a constructor. You will need to make the stack dynamically allocate the underlying array. Now you will need to provide the four basic operations: default constructor, copy constructor, destructor, and assignment operator in order to avoid having memory problems.

3. Change your class to a templated class so that your stack will work for any specified data type.

4. How might you modify your stack class to avoid overflow? That is, can you find a way to increase the size of the underlying array on the fly?

5. If you were to implement your stack class with a linked list rather than an array, could a singly linked list suffice or would you need a doubly linked list? Explain.

6. Sketch an algorithm that accepts input strings and converts them in to palindromes. For example, if hello is the input, the output should be helloolleh. Use a stack to build the palindrome.

7. We can use a stack to implement a "parentheses checker". Imagine a string that contains six special parenthesis characters, {, }, [, ], ( and ) along with other characters. We want our parenthetical strings to be balanced – that is we want opening and closing parentheses to match both in number and in type, and we want nesting to be correct as well. This idea is captured in the grammar:

```
S ---> S ( S ) S
S ---> S [ S ] S
S ---> S { S } S
S ---> B
B ---> any char not one of these six parens
```
We have an example of legal and illegal strings below:

legal:  ( ( ) ) [ { } ]
illegal  ( ) ) { }

(a) Sketch an algorithm that uses a stack to parse parenthetical strings and sort them in to legal and illegal strings.

(b) Implement a parentheses checker in a main program using your stack class to keep track of parens.

8. **Something to think about:** Consider the parentheses checking program you just wrote. How many different types of errors can show up in an expression? Can you classify them? Can you adjust your program to identify them? Can you identify the line and column number of the offending parenthesis? What happens when you input your own C++ source code in this program? Can you devise a "tree like representation" to draw for your parentheses expressions that indicates the appropriate relationships of nesting?