**Assignment 2 – Stacks**

*Write pseudo-code for problems requiring code. Do not write Java, Python or C++. You are responsible for the appropriate level of detail.*

1. **a) Use the operations push, pop, peek and empty to construct an operation which sets *i* to the bottom element of the stack, leaving the stack unchanged. (hint: use an auxiliary stack.)**

**b) Use the operations push, pop, peek and empty to construct an operation which sets *i* to the third element from the bottom of the stack. The stack may be left changed.**

1. **Simulate the action of the algorithm for checking delimiters for each of these strings by using a stack and showing the contents of the stack at each point. Do not write an algorithm.**
2. **{[A+B]-[(C-D)]**
3. **((H) \* {([J+K])})**
4. **Write an algorithm to determine whether an input character string is of the form**

***x C y***

**where *x* is a string consisting only of the letters ‘*A*’ and ‘*B*’ and *y* is the reverse of the *x* (i.e. if *x=”ABABBA”* then *y* must equal *“ABBABA”*). At each point you may read only the next character in the string, i.e. you must process the string on a left to right basis. You may not use string functions.**

1. **Write an algorithm to determine whether an input character string is of the form**

***a D b D c D … D z***

**Where each string *a, b, …z* is of the form of the string defined in problem 3. (Thus a string is in the proper form if it consists of any number of such strings from problem 3, separated by the character ‘*D*’, e.g. *ABBCBBADACADBABCBABDAABACABAA*.) At each point you may read only the next character in the string, i.e. you must process the string on a left to right basis. You may not use string functions.**

1. **Consider a language that does not have arrays but does have stacks defined as a data type. That is, one can declare**

**stack s;**

**The push, pop, empty, and peek operations are defined. Show how a one-dimensional array can be implemented by using these operations on two stacks. In particular, show how you can insert into and read from such an array.**

1. **Design a method for keeping two stacks within a single linear array s[SPACESIZE] so that neither stack overflows until all of memory is used and an entire stack is never shifted to a different location within the array. Write methods *push1, push2, pop1, and pop2* to manipulate the two stacks. (Hint: the two stacks grow toward each other.)**
2. **Transform each of the following expressions to prefix and postfix expressions. $ is exponentiation.**

**a. (A+B)\*(C$(D-E)+F)-G**

**b. A+(((B-C)\*(D-E)+F)/G)$(H-J)**

1. **Transform each of the following expressions to infix expressions.**

**a. ++A-\*$BCD/+EF\*GHI**

**b. +-$ABC\*D\*\*EFG**

**c. AB-C+DEF-+$**

**d. ABCDE-+$\*EF\*-**

1. **Apply the evaluation algorithm in the text to evaluate the following postfix expressions,**

**where A=1, B=2, and C=3.**

**a. AB+C-BA+C$-**

**b. ABC+\*CBA-+\***    

1. **Write a prefix function to accept an infix string and create the prefix form of that string, assuming that the string is read from right to left and that the prefix string is created from right to left. Handle variables, +,-,/,+,$, (,).**

**To think about and to discuss in Office Hours: Design and implement a stack in which each item on the stack is a varying number of integers. Choose a native data structure to implement your stack and design push and pop methods for it, without using library functions. Hint: Make it as simple as possible.**