National Institute of Technology, Calicut
Department of Computer Science and Engineering
CS2094 – Data Structures Lab

Assignment 4

Date of Submission: On or before 15-03-2015(Sunday) 5:00pm
(for both Advanced batch and Main batch)

Policies for Submission and Evaluation

You must submit your assignment in the moodle (eduserver) course page, on or before the submission deadline.
Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straightaway awarded F grade in the course. Detection of ANY malpractice regarding the lab course will also may lead to awarding an F grade.

Naming Conventions for submission

The source codes must be named as ASSG<Number>_<ROLLNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.<extension> (For example: ASSG4_BxxyyyyCS_LAXMAN_1.c). If there is a part *a* and a part *b* for a particular question, then, name the source files for each part separately as in ASSG4_BxxyyyyCS_LAXMAN_1b.cpp

If you do not conform to the above naming conventions, your submission might not be recognized by some automated tools, and hence will lead to a score of 0 for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

Violations of academic integrity will be severely penalized.
Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work **MUST BE** an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign **F** grade in the course. The department policy on academic integrity can be found at: http://cse.nitc.ac.in/sites/default/files/Academic-Integrity.pdf.

About the Assignment

This assignment is based on hash, binary search trees, binary trees and disjoint sets. Implement ALL questions. Your program MUST take input from text files, which is formatted as specified with each question.

Questions

1. **HASH**

Implement a hash table. The hash function to be used is the "modulo operation". Resolve collisions by using

a) Chaining.
b) Open Addressing
   i. Linear Probing
   ii. Quadratic Probing
   iii. Double Hashing (Use the multiplication method as the secondary hash function)

Your program must support the following functions:

• **insert(h, key)** – insert the data specified by key into the hash table specified by h.

• **search(h, key)** – search for the data specified by key in the hash table specified by h.

Input - Output Format

The first line contains a single positive integer c, the capacity of the hash table. All modulo operations have to be performed using c.

The rest of the input consists of multiple lines, each one containing either one or two integers.

The first integer in the line can be 0, 1, or 2, and each one has its own meaning:

- The integer 0 means stop the program.
- The integer 1 means insert the next integer from the input into the hash table. Output the index at which the data is stored. If open addressing is used, in case of a collision, output the probe sequence (here, the index at which the data will get stored must be printed only once, and at the end of the sequence).
- The integer 2 means search for the next integer from the input into the hash table. Output "FOUND", if the search is successful. Otherwise, output "NOT FOUND". If open addressing is used, output the probe sequence, before the message.

Sample Input and Output

Input	Chaining	Linear probing	Quadratic Probing
11			
2 13	2 NOT FOUND	2 NOT FOUND	2 NOT FOUND
1 45	1	1	1
1 17	6	6	6
1 29	7	7	7
1 55	0	0	0
2 28	6 NOT FOUND	6 7 8 NOT FOUND	6 7 10 NOT FOUND
1 10	10	10	10
1 21	10	10 0 1 2	10 0 3
2 21	10 FOUND	10 0 1 2 FOUND	10 0 3 FOUND
2 32	10 NOT FOUND	10 0 1 2 3 NOT FOUND	10 0 3 8 NOT FOUND
0			

Input	Double Hashing
11	
2 13	2 NOT FOUND
1 45	1
1 17	6
1 29	7
1 55	0
2 28	6 ... NOT FOUND
1 10	10
1 21	10 ...
2 21	10 ... FOUND
2 32	10 ... NOT FOUND
0	

2. BST

Create a Binary Search Tree, which supports the following operations.

- **insert(tree, element)** – adds the node specified by element (which contains the data) into the BST specified by tree.
- **search(tree, key)** – searches for the data specified by key in the BST specified by tree.
- **findMin(tree)** – retrieves the smallest data in the BST specified by tree.
- **findMax(tree)** – retrieves the largest data in the BST specified by tree.
- **predecessor(tree, element)** – retrieves the inorder-predecessor of the node specified by element in the BST specified by tree.
- **successor(tree, element)** – retrieves the inorder-successor of the node specified by element in the BST specified by tree.
- **delete(tree, element)** – removes the node specified by element from the BST specified by tree.
- **inorder(tree)** – To do a recursive inorder traversal of the BST.
- **preorder(tree)** – To do a recursive preorder traversal of the BST.
- **postorder(tree)** – To do a recursive postorder traversal of the BST.

Input - Output Format

The input consists of multiple lines, each one containing either one or two integers.

The first integer in the line can be 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 or 10 and each one has its own meaning:

- The integer 0 means stop the program.
- The integer 1 means, create a node which contains the next integer from the input as the data part, and then, insert this node into the BST. In this case, the next integer (>= 0) is given on the same line as the 1, separated by a space.
- The integer 2 means, search for the key specified by the next integer from the input, in the BST. In this case, the next integer (>= 0) is given on the same line as the 2, separated by a space. If the search is successful, output "FOUND". Otherwise, output "NOT FOUND".

- The integer 3 means find and output the minimum number in the BST. Print “NIL” if the BST is empty.
- The integer 4 means find and output the maximum number in the BST. Print “NIL” if the BST is empty.
- The integer 5 means find and output the inorder-predecessor of the data specified by the next integer from the input. In this case, the next integer (>= 0) is given on the same line as the 5, separated by a space. Output “NIL”, if the data exists in the tree, but there is no inorder-predecessor for the data. Output “NOT FOUND”, if the data is not present in the tree.
- The integer 6 means find and output the inorder-successor of the data specified by the next integer from the input. In this case, the next integer (>= 0) is given on the same line as the 6, separated by a space. Output “NIL”, if the data exists in the tree, but there is no inorder-successor for the data. Output “NOT FOUND”, if the data is not present in the tree.
- The integer 7 means delete the data specified by the next integer in the input from the BST, if present. In this case, the next integer (>= 0) is given on the same line as the 7, separated by a space. (Here, the data to be deleted is guaranteed to be present in the BST).
- The integer 8 means output the in-order traversal of the BST.
- The integer 9 means output the pre-order traversal of the BST.
- The integer 10 means output the post-order traversal of the BST.

```
Sample Input                         Sample Output

2 25                                 NOT FOUND
3                                    NIL
4                                    NIL
5 25                                 NOT FOUND
6 25                                 NOT FOUND
1 25
2 25                                 FOUND
3                                    25
4                                    25
5 25                                 NIL
6 25                                 NIL
1 13
1 50
1 45
1 55
1 18
2 10                                 NOT FOUND
2 13                                 FOUND
2 35                                 NOT FOUND
2 55                                 FOUND
2 80                                 NOT FOUND
8                                    13 18 25 45 50 55
9                                    25 13 18 50 45 55
```

```
10                        18 13 45 55 50 25
3                         13
4                         55
5 13                      NIL
6 13                      18
5 18                      13
6 18                      25
5 25                      18
6 25                      45
5 45                      25
6 45                      50
5 50                      45
6 50                      55
5 55                      50
6 55                      NIL
7 55
7 13
7 25
3                         18
4                         50
5 18                      NIL
6 18                      45
5 45                      18
6 45                      50
5 50                      45
6 50                      NIL
7 45
7 50
7 18
3                         NIL
4                         NIL
1 90
3                         90
4                         90
0
```

3. EXPTREE

Given a valid postfix expression, create the corresponding expression tree. Traverse this tree in inorder, preorder and postorder fashion (all three methods, both recursively and iteratively).
The program must support (at least) the following in the postfix expression:

Binary operators:

^	: Exponentiation (Highest precedence)
/, *	: Division, Multiplication
+, -	: Addition, Subtraction(Lowest precedence)

Operands:

The operands are all variables, which is represented by a single lowercase English character. (a-z)

Input-Output Format

- The only line of the input contains a valid postfix expression. (There will not be any space anywhere in the expression)
- Do the inorder, preorder and postorder (both recursively and iteratively). Output the result of each traversal on a fresh line.
- For the inorder traversals, use proper parentheses also (i.e., each operator, together with its operands must be enclosed in parentheses).

Sample Input/Output

```
abc*+
                    Inorder:   (a+(b*c))
                    Preorder:  +a*bc
                    Postorder:  abc*+

m
                    Inorder: m
                    Preorder:  m
                    Postorder: m

abcd^*+e-
                    Inorder:   ((a+(b*(c^d)))-e)
                    Preorder:   -+a*b^cde
                    Postorder: abcd^*+e-
```

****