National Institute of Technology, Calicut
Department of Computer Science and Engineering
CS2094 – Data Structures Lab
Assignment-4

Submission deadline (on or before):
29th February 2016, 10:00:00 PM (for both Main and Advanced batches)

Policies for Submission and Evaluation

You must submit your assignment in the moodle (Eduserver) course page, on or before the submission deadline. Also, ensure that your programs in the assignment must compile and execute without errors in Athena server. During evaluation your uploaded programs will be checked in Athena server only. Failure to execute programs in the assignment without compilation errors may lead to zero marks for that program.

Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straight away awarded zero marks for this assignment and considered by the examiner for awarding F grade in the course. Detection of ANY malpractice regarding the lab course will also lead to awarding an F grade.

Naming Conventions for Submission

Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar or .tar.gz). The name of this file must be ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>.zip (For example: ASSG3_BxxyyyyCS_LAXMAN.zip). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

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.c.

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.

Checking your assignment with sample test cases:

You must check your programs in this assignment with the sample input test case files that are available in the lab course page. Also, verify your output with the sample output files. Your programs will be evaluated using the script which compiles all your programs and verify your output with the correct output. Hence, do not print any extra messages while reading the input from the user as well as printing the output. You must refer the sample input and output files for this.

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.

Assignment Questions

General Instructions for all the questions:

- Invalid input should be detected and suitable error messages should be generated.
- Sample inputs are just indicative.

1. Implement a hash table. The hash function to be used is the “modulo operation”. Resolve collisions by using
   a. Chaining.
   b. Open Addressing
      - Linear Probing(take “modulo operation” for auxiliary hash function also)
      - Quadratic Probing(take c1=0 and c2=1 as auxiliary constants and “modulo operation” for auxiliary hash function)
      - Double Hashing (Use the multiplication method as the secondary hash function with constant A= 0.6180339887)

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 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 a character followed by zero or one integer. The meaning of each line is given below:

- Character 's' means stop the program.
- Character 'i' 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). If the data cannot be inserted, then print “CANNOT INSERT”.
- Character 'f' means search for the next integer from the input in 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.

Output format:

- The output (if any) of each command should be printed on a separate line.

Sample Input and Output

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

i 10	10	10	10
i 21	10	10 0 1 2	10 0 3
f 21	10 FOUND	10 0 1 2 FOUND	10 0 3 FOUND
f 32	10 NOT FOUND	10 0 1 2 3 NOT FOUND	10 0 3 8 NOT FOUND
s			

Input	Double Hashing
11	
f 13	2 NOT FOUND
i 45	1
i 17	6
i 29	7
i 55	0
f 28	6 9 NOT FOUND
i 10	10
i 21	10 9
f 21	10 9 FOUND
f 32	10 4 NOT FOUND
s	

2. 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 format:

The input consists of multiple lines, each containing a four letter string followed by zero or one integer. The meaning of each line is given below:

- String “stop” means stop the program.

- String “insr” means, create a node which contains the next integer(>= 0) from the input as the data part, and then, insert this node into the BST. In this case, the data is given on the same line as the string “insr”, separated by a space.

- String “srch” means, search for the key specified by the next integer(>= 0) from the input, in the BST. In this case, the key is given on the same line as the string “srch”, separated by a space. If the search is successful, output “FOUND”. Otherwise, output “NOT FOUND”.

- String “minm” means find and output the minimum value in the BST. Print “NIL” if the BST is empty.

- String “maxm” means find and output the maximum value in the BST. Print “NIL” if the BST is empty.

- String “pred” means find and output the inorder-predecessor of the data specified by the next integer (>= 0) from the input. In this case, the data is given on the same line as the string “pred”, 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.

- String “succ” means find and output the inorder-successor of the data specified by the next integer(>= 0) from the input. In this case, the data is given on the same line as the string “succ”, 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.

- String “delt” means delete the data specified by the next integer(>= 0) in the input from the BST, if present. In this case, the data is given on the same line as the string “delt”, separated by a space. (Here, the data to be deleted is guaranteed to be present in the BST).

- String “inor” means output the in-order traversal of the BST.

- String “prer” means output the pre-order traversal of the BST.

- String “post” means output the post-order traversal of the BST.

Output format:

- The output (if any) of each command should be printed on a separate line.

Sample Input	Sample Output
srch 25	NOT FOUND
minm	NIL
maxm	NIL
pred 25	NOT FOUND

succ 25	NOT FOUND
insr 25	
srch 25	FOUND
minm	25
maxm	25
pred 25	NIL
succ 25	NIL
insr 13	
insr 50	
insr 45	
insr 55	
insr 18	
srch 10	NOT FOUND
srch 13	FOUND
srch 35	NOT FOUND
srch 55	FOUND
srch 80	NOT FOUND
inor	13 18 25 45 50 55
prer	25 13 18 50 45 55
post	18 13 45 55 50 25
minm	13
maxm	55
pred 13	NIL
succ 13	18
pred 18	13
succ 18	25
pred 25	18
succ 25	45

pred 45	25
succ 45	50
pred 50	45
succ 50	55
pred 55	50
succ 55	NIL
delt 55	
delt 13	
delt 25	
minm	18
maxm	50
pred 18	NIL
succ 18	45
pred 45	18
succ 45	50
pred 50	45
succ 50	NIL
delt 45	
delt 50	
delt 18	
minm	NIL
maxm	NIL
insr 90	
minm	90
maxm	90
stop	

3. Given a valid prefix expression, create the corresponding expression tree, and implement the following:

a) Traverse the tree recursively and print preorder, postorder and inorder traversals.
b) Traverse the tree iteratively and print preorder, postorder and inorder traversals.

The program must support (at least) the following in the prefix 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 Format
• The only line of the input contains a valid prefix expression. (There will not be any space anywhere in the expression)

Output Format
• The inorder, preorder and postorder (both recursively and iteratively). Output the result of each traversal on a fresh line.
• For the inorder traversal, use proper parentheses also (i.e., each operator, together with its operands must be enclosed in parentheses).

The following sample input and output is for both the recursive(3a) and iterative(3b) traversals.

Sample Input	*Sample Output*
Example 1	
+a*bc	Inorder: (a+(b*c)) Preorder: +a*bc Postorder: abc*+
Example 2	
m	Inorder: m Preorder: m Postorder: m
Example 3	
-+a*b^cde	Inorder: ((a+(b*(c^d)))-e) Preorder: -+a*b^cde Postorder: abcd^*+e-

****