Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-717 Advanced Computing for Electrical Engineers

Assignment 2

Submission Deadline: August 28, 2014 (Thursday), 11:30 pm (IST)

There are minor edits made after uploading on Moodle. They are marked in RED. While the edits to the document are mostly for better clarity, there are some important changes. Please read carefully.

Note: You can use either C or C++ to do this assignment. You must follow academic ethics. Plagiarism is punishable. Any academic unethical act may result in a FAIL grade.

The goal of this assignment is to get you to implement basic data structures and to evaluate their performance.

Problem 1:

Write a program that will implement two stacks S0 and S1 using a single array. Each stack should support the basic operations 1) Push 2) Pop 3) Peek and 4)IsEmpty. The program should read from primary input and write results to screen.

Input format:
The first line contains N, an integer, which gives the size of the array. This is the total size of both the stacks combined.

From second line onwards, we specify the operations on the stack in the following format:

<operation id> <stack num> <optional operand>

where <operation> is an integer in 0..3
0 is for push()
1 is for pop()
2 is for peek()

3 is for isEmpty()
<stack num> is either 0 or 1 identifying the stack to be operated on
<optional operand> is applicable only for push(). It takes a positive integer as the element to be pushed.

Output: For each operation, you will have to print one line of output following the format below:

<result>

For push(), the result must be 1 if it successful, -1 if it is not (could happen if the stack is full).

For pop(), the result must pop the element at the top of the stack and return that if successful. If the stack is empty when pop() is called, the result must be -1.

For peek(), the result must be the element on the top of the stack if successful. If the stack is empty, the result must be -1.

For isEmpty(), the result is 1 if it is empty and 0 otherwise.

Constraints:

The size of the array combined is restricted to 1<= N <= 10000.

The elements to be pushed in both the stacks are positive integers (>0).

There is no restriction on the number of operations on each stack.

There is no restriction on how big each of the stacks can grow. Only the combined size is restricted by N.

You do not have to do any sanity check on the value of N or on the integer elements that are to be pushed. We will guarantee that there will be no invalid inputs.

Example:

Input	Expected output
6 1 0 1 1 0 0 51 0 1 31 0 0 52 0 1 32 0 0 53 0 1 33 1 0 1 1 2 0 2 1 0 0 54 0 1 34 0 0 55 2 0 1 0 1 0 3 0 3 1 2 1	-1 -1 1 1 1 1 1 53 33 52 32 1 1 -1 54 54 52 0 0 34

Problem 2:

Write a program that will implement Hash data structure for strings. We will use a popular Hash function called the Jenkin's One-At-A-Time hash function. This hash function is very fast and has several interesting properties (including what is called

the avalanche effect).

The hash function is presented in a C snippet below. If you include *stdint.h,* you will get access to the uint32_t datatype.

```
uint32_t jenkins_one_at_a_time_hash(char *key)

{

    uint32_t hash, i;



    int len = strlen(key);

    for(hash = i = 0; i < len; ++i)

    {

        hash += key[i];

        hash += (hash << 10);

        hash ^= (hash >> 6);

    }

    hash += (hash << 3);

    hash ^= (hash >> 11);

    hash += (hash << 15);

    return hash;

}
```

The function takes a single NULL terminated string ***key*** as input and returns a unsinged 32-bit integer as output.

Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-717 Advanced Computing for Electrical Engineers

For this problem, you will be peforming operations on strings. Each string will be utmost 19 characters in length (20 characters including the NULL termination character).

You have to use linear probing with f(i) = i and $h_i(x)$ = (h(x)+f(i)) where *i* indicates the probe number starting from 0.

The deletion algorithm should use *Lazy Deletion* with values -2 for EMPTY and -1 for FREE.

INPUT:

The first line of the input file will contain the size of the hash table (M). This will be followed by one or more pairs of input of the following format:
<Operation id> <String>

where operation id is 0 – if the string has to be searched
1 – if the string has to be inserted and
2 – if the string has to be deleted.
3 – to print the table

The string will be a sequence of characters containing A-Z, a-z and 0-9.

During insertion, this string has to be stored at the appropriate position.

OUTPUT:

The output will be a sequence of integers, one integer per line for each operation.

If the operation is search(),
print -1 for the case where key is not found
print position if the key is found (even if the values don't match)

If the operation is insert(),
print -1 if the hash table is full
print position where the key was inserted otherwise

If the operation is delete(),

print -1 if the key is not found
print -2 if you found some other entry with the same key
print position where the key was deleted from otherwise

If the operation is print,
call the printTable function.

CONSTRAINTS:

The size of the hash table will be 1 <= M <= 10,000. The array used for hashtable should have indices in the range 0 to M-1.

The input strings need not be validated. You can assume that they are of appropriate length and contains only valid characters as specified earlier.

When a string is input, you locate the correct position for storing it. Store the key in the array called hashTable and the string in the same index in the array called valTable.

It is possible that, when you search for a string S1, you encounter a position in which the key stored is same as the key that you are searching for but the string that is saved there is some other string S2. Even if that is the case, search is considered successful. Return the position. This is a spurious search!

It is possible that when you delete, a similar situation arises. You want to delete a value V1 with key K but you encounter a position where the key happens to be K, but the value is some V2 and not V1. In such a case, do not delete the (key, value) pair. The delete is deemed unsuccessful and you have to return -2. Note that because of the probe sequence, it is possible that the key-value pair (V1,K) could be further down the sequence but you located the key K ahead because of V2. If such a case happens, do not probe further and return as unsuccessful delete. This is a spurious delete – even though the key is there, you are returning back unsuccessfully.

For this problem, you are also given a template file called hashTemplate.c
There is a portion called the prefix code and there is another portion in it called the suffix code. These portions have to be retained as it is. You can make changes only in the portions between these two lines. FOLLOW THIS STRICTLY. YOUR PROGRAM WILL NOT BE EVALUATED IF YOU VIOLATE THIS.

Indian Institute of Technology Bombay
Department of Electrical Engineering
EE-717 Advanced Computing for Electrical Engineers

4. How to submit?

1. A single .zip file is uploaded on the website with the name rollno.zip. Download and unzip the file. Change the directory called rollno to your roll number. The directory name should be all lower case.

2. Under the rollno directory, you will see two directories, prob1 and prob2 for the two problems.

3. Empty file called “stack.c” is provided under prob1 with the appropriate Makefile.

4. Empty file called “hash.c” is provided under prob2 with the appropriate Makefile. That directory also contains a file called “hashTemplate.c” which you should use for your program. There is also a test file with the name input.txt in that directory.

5. Make changes to stack.c and hash.c and finish your work.

6. Under the rollno directory, there is a README.txt file. Put your name and roll number there. If you want to put any comments there for us to see, do so.

7. When you are satisified with the work, run make clean in both prob1 and prob2 directories separately. This will delete all the temporary files, input files, output files etc. Only the Makefile and a .c file should remain in each directory.

8. Go back to your rollno folder and zip everything including the directory. This should create a .zip file called “rollno.zip”.

9. Possibly send this file as an email to yourself and check if you are able to download and extract the files succsefully.

10. Upload this .zip file on Moodle.

Changes to .zip file that is uploaded:

There are some minor changes to the .zip file.

- 1. The Makefile that was supplied had a minor error. It is fixed now.
- There are three changes made to hashTemplate.c in line numbers 57, 58 and 82. The changes are minor but the search/insert/delete functions should be changed appropriately. Also, this change will result in the output files being different than any output files that you wrote previously.