Computer Engineering Department

TED University

CMPE 252 C Programming, Spring 2020

TERM PROJECT

26.05.2020, Tuesday

DUE DATE: 16.06.2020, Tuesday, 23.59

Send your C code file via Moodle.

This is an individual work. No team work is allowed.

Similarity check will be applied to submitted codes.

NETWORK STRUCTURE CONSTRUCTION & MAINTENANCE TOOL

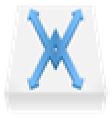
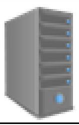
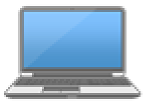
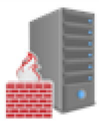
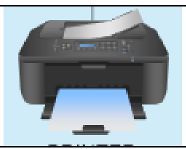
In the term project, you are expected to use your C programming skills on subjects such as:

- Pointers and Dynamic Data Structures
- Recursion
- Personal Libraries

1 PROBLEM

The project consists of constructing and maintaining the network structure of an enterprise using trees.

Figure 1 shows the network structure of a sample enterprise which includes several types of nodes connected hierarchically. The possible list of nodes in the network are:

NODE TYPE	DEPICTION		NODE TYPE	DEPICTION
SWITCH			WS (Workstation)	
ROUTER			TERMINAL	
INTERNET			IMAC	
SERVER			LAPTOP	
FIREWALL			PRINTER	

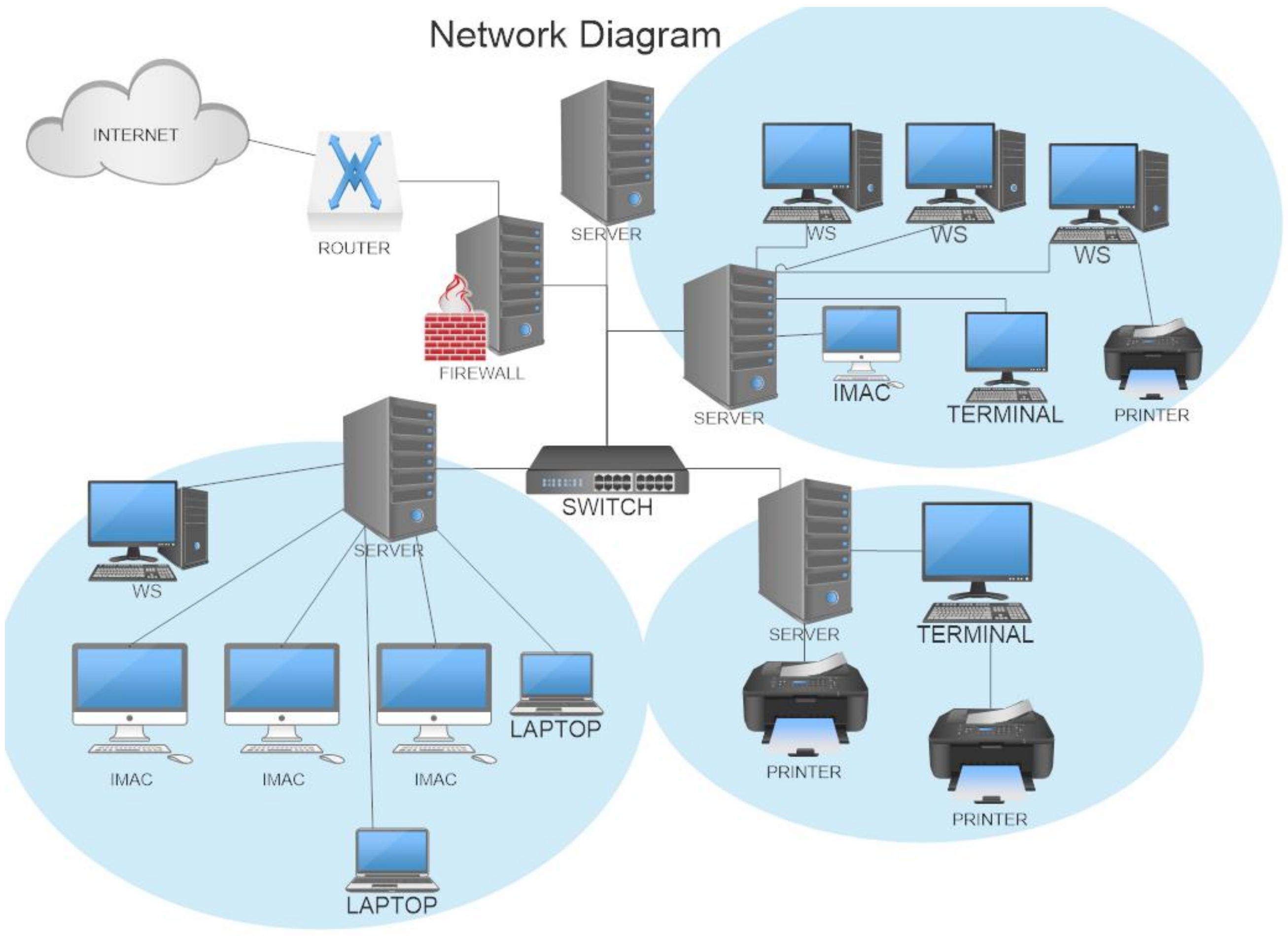


Figure 1 : Network Diagram of LAN of a Sample Enterprise

The network structure is a dynamic structure in a tree shape, where nodes can not be connected in cyclic form and they should be connected hierarchically. The number of nodes that can be connected to a parent node in the network tree is limited by the number of available hardware interfaces (you can assume, maximum number of children of a parent node <=10).

1.1 CONSTRUCTING A NETWORK TREE (INSERTING NODES)

The network structure will be constructed using an input script language which has separate commands in each line of the script file (which is a text file). When constructing the nodes in the tree form, each node will be assigned to a Unique ID which will be determined by maintaining a counter variable starting from "1" to the number of nodes that are inserted until that insertion operation and incremented at each insertion of a new node.

The below table shows list of commands that is used to "construct" a tree:

Table 1 : Tree Construction Commands

INSERT <*node*>	This is always the first line of the input commands. It creates a new tree with the root node given in <*node*>. A network can have only one root and its Unique ID is always set to 1. e.g. "INSERT SWITCH" creates a tree with root SWITCH with ID=1.

INSERT *<node>* *<parent>*[*<ID>*]	*This command adds child node to the parent node.* *<node>* and *<parent>* are strings providing the corresponding node types to be added to the network. The last term given in brackets is the Unique ID of the parent node, this is because there can be other nodes of the same type with the given parent node type. e.g. "INSERT FIREWALL SWITCH[1]" command creates a new node of type FIREWALL and it is added as a child node of the SWITCH node whose ID = 1. If this is the second insertion so far, then the ID of the FIREWALL will be set to 2. e.g.-2: Next, if another command appears in the script on the next line such as: "INSERT ROUTER FIREWALL[2]" , then this command creates a node of type ROUTER with ID=3 and adds it to FIREWALL node with ID = 2 as its child node.

The below sample input script file which has separate commands in each line is used to construct the network tree of the diagram depicted in Figure 1.

```
INSERT SWITCH [ROOT]
INSERT FIREWALL SWITCH[1]
INSERT ROUTER FIREWALL[2]
INSERT INTERNET ROUTER[3]
INSERT SERVER SWITCH[1]
INSERT SERVER SWITCH[1]
INSERT SERVER SWITCH[1]
INSERT WS SERVER[5]
INSERT TERMINAL SERVER[6]
INSERT IMAC SERVER[5]
INSERT IMAC SERVER[5]
INSERT IMAC SERVER[5]
INSERT LAPTOP SERVER[5]
INSERT PRINTER TERMINAL[9]
INSERT PRINTER SERVER[6]
INSERT LAPTOP SERVER[5]
INSERT WS SERVER[7]
INSERT WS SERVER[7]
INSERT WS SERVER[7]
INSERT IMAC SERVER[7]
INSERT TERMINAL SERVER[7]
INSERT PRINTER WS[19]
```

!! Note that insertions occur at the child links of the parent nodes from left to right and the first empty link is used for establishing a new link to insert a child node.

The constructed tree according to above commands are depicted in below figure (Figure 2):

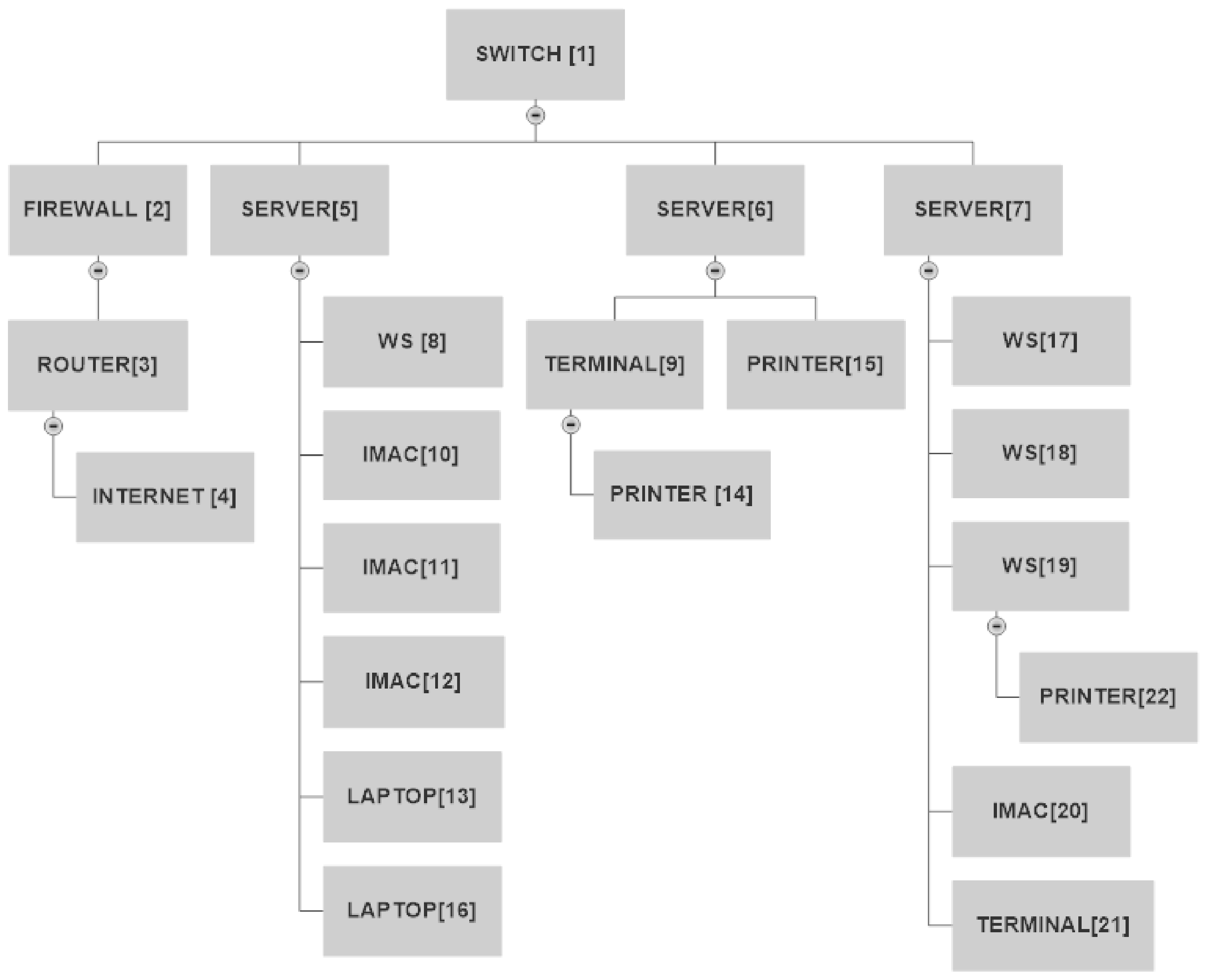


Figure 2 : Constructed Tree of Sample Network

1.2 REMOVING NODES FROM THE NETWORK (DELETING NODES)

Network structures are dynamic in nature where one or more nodes can go power off or disconnected from the network. If the removed node is the root node of a subnetwork (subtree), other nodes in that subtree will not be accessible any more and therefore, they should also be removed from the network.

Therefore, below command in the input script file enables removing a node and all of its subtree if applicable.

DELETE <*node*>[<*ID*>]	This command removes the node from the network tree and if it is a parent node (with children) then all the subtree nodes will also be removed from the network tree.

	If it is the root node, all the network will be deleted. e.g. “DELETE FIREWALL[2]” command removes all the leftmost subtree of SWITCH[1] root node in the sample network in Figure 1. e.g.-2: “DELETE PRINTER[22]” will only remove the leaf node whose parent is WS[19]. e.g.-3: After above deletions if another insertion occurs such as “INSERT LAPTOP WS[19]”, this command will create a new LAPTOP node which will be the only child node of WS[19] and it will take the Unique ID = 23 which is the next value of the insertion counter maintained.

!! Deletion does not affect the Unique ID’s of the nodes in the network. If, afterwards, new insertions happen, they will get IDs from the main counter incremented by the latest ID used.

Below figure (Figure 3) shows the depiction of the resulting tree structure of the network tree after below list of commands are executed on the current network tree depicted in Figure 2:

```
DELETE FIREWALL[2]
DELETE PRINTER[22]
INSERT LAPTOP WS[19]
INSERT LAPTOP SWITCH[1]
```

You may notice in Figure 3 that:

- FIREWALL[2] which is the first child node (leftmost) of SWITCH[1] root node is deleted along with all of its subtree by the 1st command line “DELETE FIREWALL[2]”
- 2nd command line deleted PRINTER[22] under WS[19].
- 3rd command line inserted a new node of type LAPTOP under WS[19] which has no childs after previous deletion. Note that Unique ID of the LAPTOP node determined as 23 since the latest insertion ID was 22.
- 4th command line inserted a new node under SWITCH[1], to the leftmost child link of this parent node, since that child link was empty after deletion of FIREWALL[2]. (Therefore, if due to deletions, child links become NULL, new insertions will use the first available empty (NULL) link while scanning from left to right child links.

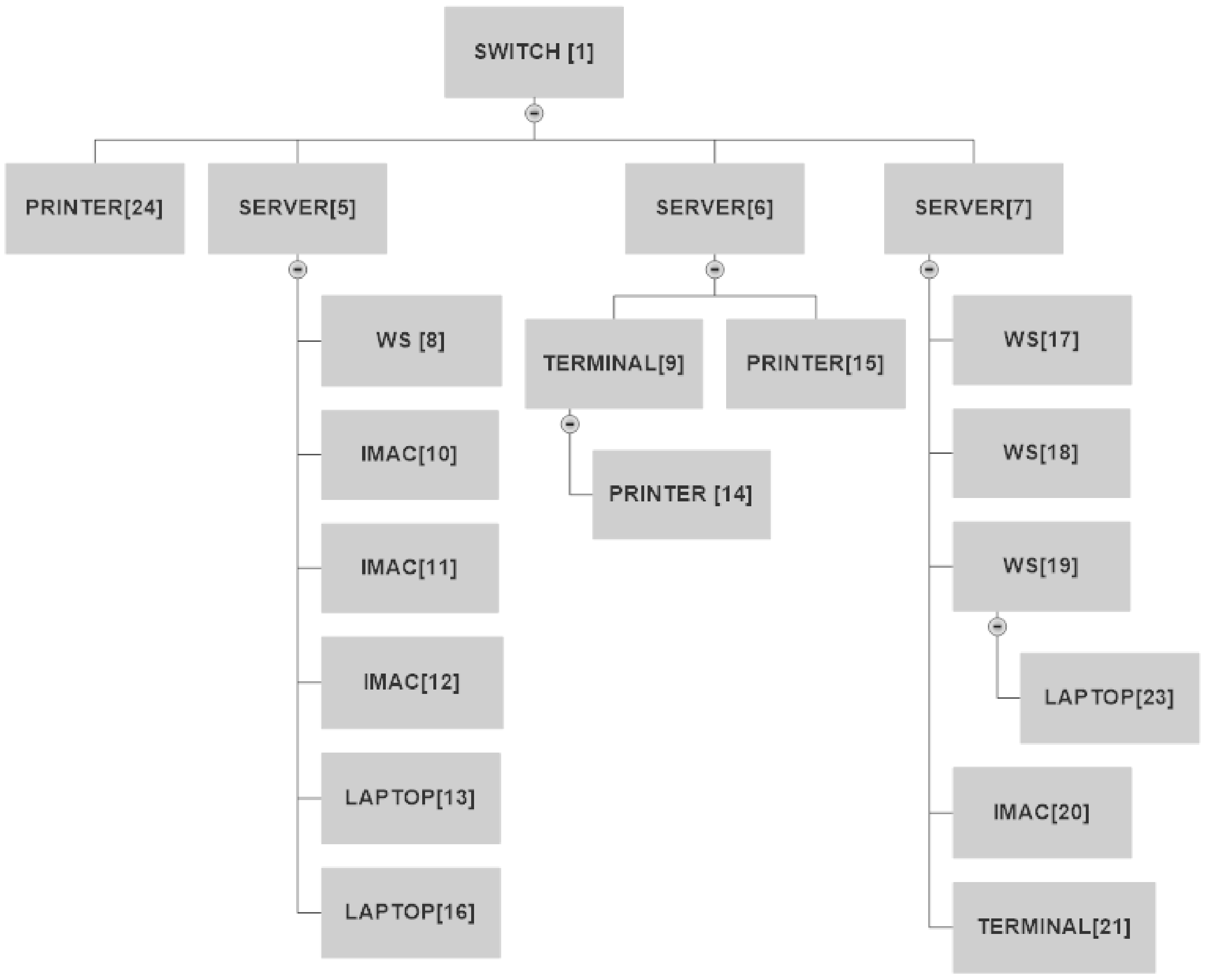


Figure 3: New Tree Structure after deletions and insertion.

1.3 PRINTING THE NETWORK TREE

Due to the dynamic structure of the network, we often need to understand the current structure of the network.

This will be performed by the "PRINT" command with the given rules below:

PRINT *<node>[<ID>]*	This command prints the tree or subtree starting with the given node in PREORDER traversal mode. PREORDER traversal mode of an N-ary tree will follow below rules: - *Visit current node* - *Visit leftmost subtree (1st child link from left to right)* - *Visit second leftmost subtree. (2nd child link from left to right)* - *Visit third leftmost subtree. (3rd child link)*

	- *..* - *Visit Mth leftmost subtree (Mth child link)* - *..* - *Visit rightmost subtree (last child link of the parent node)*

For example, “PRINT SWITCH[1]” and “PRINT SERVER[6]” commands for the tree given in Figure 3 will print to the output file the below output lines of strings corresponding to PREORDER visited nodes:

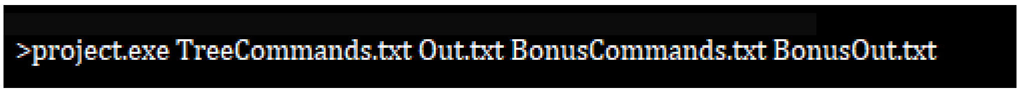
Output of “PRINT SWITCH[1]” command	**Output of “PRINT SERVER[6]” command**
SWITCH[1] PRINTER[24] SERVER[5] WS[8] IMAC[10] IMAC[11] IMAC[12] LAPTOP[13] LAPTOP[16] SERVER[6] TERMINAL[9] PRINTER[14] PRINTER[15] SERVER[7] WS[17] WS[18] WS[19] LAPTOP[23] IMAC[20] TERMINAL[21]	SERVER[6] TERMINAL[9] PRINTER[14] PRINTER[15]

2 REQUIREMENTS

1. **Your source code shall consist of 3 files:**
   - **main.c : includes only the main function that processes the input arguments and calls tree library functions.**
   - **tree.c: constitutes your personal library including your own tree constructing, maintenance and printing functions.**
   - **tree.h: header file of your tree personal library.**

2. **Your program shall be executed from the command line as below using 4 command line arguments:**

>program.exe <*InputCommandsScriptFilename*> <*OutputFilename*> <*BonusInputScriptFilename*> <*BonusOutputFilename*>



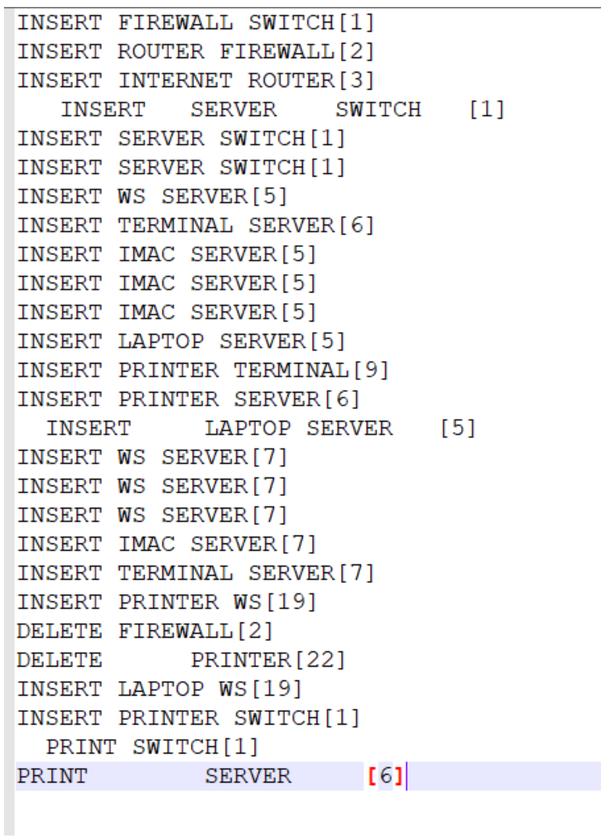
```
>project.exe TreeCommands.txt Out.txt BonusCommands.txt BonusOut.txt
```

where, each command line argument is described as below:

<*InputCommandsScriptFilename*> : The input commands script filename which includes commands for constructing and maintaining the network tree due to the dynamic structure of the network. The main rules regarding the format of the commands script file are:

- **Each line has a separate command and the commands should be executed line by line.**
- **There will not be any empty or unrecognized / unknown command line.(You can assume that the input is error free)**
- **The whitespaces in a command line should be simply ignored / skipped, a word in the command line can be followed by any number of whitespace characters.**
  - **e.g. "INSERT SERVER SWITCH[1]" and "   INSERT   SERVER   SWITCH   [1]" should be interpreted as the same.**
- **"INSERT <node>" command will be in the first line of the script and will not be repeated in any other line (there can be no two separate trees, nor a deleted tree can not be recreated)**
- **Except for the first "INSERT <node>" statement, the INSERT , DELETE, PRINT command lines can be in any order, as long as the script constructs and maintains a valid tree.**

Below is a valid input command script file opened in a text editor:



```
INSERT FIREWALL SWITCH[1]
INSERT ROUTER FIREWALL[2]
INSERT INTERNET ROUTER[3]
   INSERT   SERVER    SWITCH   [1]
INSERT SERVER SWITCH[1]
INSERT SERVER SWITCH[1]
INSERT WS SERVER[5]
INSERT TERMINAL SERVER[6]
INSERT IMAC SERVER[5]
INSERT IMAC SERVER[5]
INSERT IMAC SERVER[5]
INSERT LAPTOP SERVER[5]
INSERT PRINTER TERMINAL[9]
INSERT PRINTER SERVER[6]
  INSERT     LAPTOP SERVER   [5]
INSERT WS SERVER[7]
INSERT WS SERVER[7]
INSERT WS SERVER[7]
INSERT IMAC SERVER[7]
INSERT TERMINAL SERVER[7]
INSERT PRINTER WS[19]
DELETE FIREWALL[2]
DELETE      PRINTER[22]
INSERT LAPTOP WS[19]
INSERT PRINTER SWITCH[1]
  PRINT SWITCH[1]
PRINT        SERVER      [6]
```

<*OutputFilename*> : The output file name which includes the output of the PRINT statements processed in the order of appearance in the input command script file.

<BonusInputScriptFilename> <BonusOutputFilename> : These are the bonus part input script filename and bonus part output filename which are described in another file “TermProject-BONUSPART.pdf”.

!!Even if you did not implemented the bonus part, your executable should accept 4 arguments, but just discard / ignore the bonus input script file and don’t generate the bonus output file.

3. **Along with this TERM PROJECT description, you will be provided;**
   - 5 sample debug case input / output files:
   (“Inp1.txt Out1.txt”, “Inp2.txt Out2.txt”, “Inp3.txt Out3.txt”, “Inp4.txt Out4.txt”, “Inp5.txt Out5.txt”)
   - 5 sample bonus part debug case input / output files : (“BonusInp1.txt BonusOut1.txt”, “BonusInp2.txt BonusOut2.txt”, “BonusInp3.txt BonusOut3.txt”, “BonusInp4.txt BonusOut4.txt”, “BonusInp5.txt BonusOut5.txt”)

which are generated for your own debugging purposes. Please make sure that your output files are “exactly the same” as these output files corresponding to below sample execution of our ground truth software so that you will be sure that your solution and the output format is correct.

```
TermProject.exe Inp1.txt Out1.txt BonusInp1.txt BonusOut1.txt
TermProject.exe Inp2.txt Out2.txt BonusInp2.txt BonusOut2.txt
TermProject.exe Inp3.txt Out3.txt BonusInp3.txt BonusOut3.txt
TermProject.exe Inp4.txt Out4.txt BonusInp4.txt BonusOut4.txt
TermProject.exe Inp5.txt Out5.txt BonusInp5.txt BonusOut5.txt
```

3 GRADING

Grading will consist of 3 parts:

PART A: (15 pts)

For grading, your executables will be executed using ;

- **5 debug test case parameters given along with the project description,**
- **5 more other interesting test cases that will not be provided to you, by using different input scripts <u>which has no number of commands limitation requirement,</u>**

and exactly the same output with the ground truth solution will be expected from your software.

PART B: (5 pts)

- **Instructor Source Code Review:**
    - **Your source code will be reviewed by your instructors by evaluating your source code regarding;**
        - **Proper usage of pointers and dynamic data structures,**
        - **Proper usage of recursion,**
        - **Proper usage of personal libraries,**
        - **Nice and understandable variable, argument and function names,**
        - **Nicely commented source code files as suggested / described by your book, where each function, variable and arguments are described properly**, *(refer to Chapter 2, Chapter 3 and 12 in your book for comment styles)*
            - ***Regarding functions, you should provide function comment headers explaining the purpose, precondition, post-conditions (outputs) and explanation of parameters***
        - **No possible memory leaks by freeing the allocated space properly once that memory space is no longer needed.**
        - **Possible error conditions are checked properly, e.g. pointers and returned result of malloc statements are checked against NULL value.**

- **PART C: (7 pts)**
    - **BONUS PART outputs will be evaluated as described in bonus part description file “TermProject-BonusPart.pdf”.**