

Name: Mushfiq Rashid

Title: Program 5 Report

Course: CS 3340.002

Date: 17<sup>th</sup> November 2019

## Hex Calculator

### I. Problem Statement

The assigned task was to design a program that simulates the operation of the Microsoft calculator in programmer mode using hex input and 32-bit element size. The program must request the user for a file name in path directory format that will include hexadecimal values (one per line). Upon receiving the name, the program will open the file, read the contents, then convert the values to decimal format, storing it into internal storage. Then, the operands and operators from the file should be used to execute mathematical operations. The supported operations are + (addition), - (subtraction), \* (multiplication), / (division), % (modulus), = (assignment, equals), s (memory save), r (memory recall), and z (memory clear, zero) where there is one and only one memory location.

### II. Approach to Solution

The decision was made to generate a program which could be used by any user to enter any file name including all kinds of mathematical operations and execute the mathematical operations according to the proper precedence. The program was written in the MIPS Assembly Language using MARS as the IDE. The first part is the *.data* which includes all the variables to be used, while the second part contains the *.text* which contains the procedures (in this case, the main procedure, *hexaConversion*, *loopNewLine*, and *calc*). In order to test for the reliability and accuracy of the program, several different files were used, and the output in console was matched against the contents' actual decimal value in each file and the output that would have been produced had it been executed in Microsoft Calculator app. The following test cases were used:

1. When the operands and operators in the text file are all valid including memory operators like r (memory recall), s (memory save).
2. When the sample input text file is used.
3. When the operands and operators in the text file are all valid mathematical operations without any memory operator like r (memory recall), s (memory save), or z (memory clear).
4. When there is an error in the operators to verify error handling.

### III. Solution Description

The assigned task was to design a program that requests the file name from the user and then reads the file to find the hexadecimal values and outputs the result of the mathematical operations performed on the converted values to the console. The program was made effective by using instructions like load immediate (*li*), load address (*la*), *op* (for operations), *print* (for printing). The *li* instruction was used to specify commands like taking input of the user's file name in string form and printing the prompt message. It was also used to specify different operations like opening, reading, and closing the file. The *la* instruction was used to load addresses of the variables like *prompt*, which is the prompt message asking for the file name, and *userFileName*, which is the string containing the user's file name in directory format. The *op* keyword was used for addition, subtraction, multiplication, division, modulo operations on the values stored in the memory registers.

The *hexaConversion* function was used to convert each hexadecimal value into decimal value. It was made possible using few loop methods inside it. It also checks for invalid cases, like when the number has lowercase letters, or negative integers.

The *calc* function was used to operate on the mathematical operands and operators stored in the registers from the text file, and then printing them onto the screen.

The code to the program was newly written by the author, Mushfiq Rashid, while some previously written code written by Mushfiq Rashid was reused and modified in this program. However, and no external code from outside sources was used. The program was built and assembled in the MARS IDE where the operation was completed successfully. Figure 6 represents the execution screen after it was built successfully. The program was tested for different hexadecimal values in different files with the assumption that all values were in fixed format. The outputs of the execution for different inputs show that:

1. The program runs successfully when the user file, *input1.txt* shown in *figure 7*, has different valid hexadecimal values and memory operators. This is verified by the output screen displaying the correct results in *figure 1*, and then matched with the Microsoft Calculator app performing the same operations and outputting the same result, as shown in *figure 2*.
2. The program runs successfully when the sample input file, *input.txt* shown in *figure 8*, was used as it outputs the correct values shown in *figure 3*. This was matched against the sample output file, shown in *figure 11*, which verified that the output was successful.
3. The program runs successfully when an input file with simpler valid operations without any memory operators, as shown in *figure 9*, as it produces the correct results displayed in *figure 4*.
4. Error handling is executed without any errors, as input file containing invalid hexadecimal values, shown in *figure 10*, is ran in the program and the output converts to 0 for all the invalid values, in order to avoid invalid operations in the program. This is shown in *figure 5* where the operation results to 0.

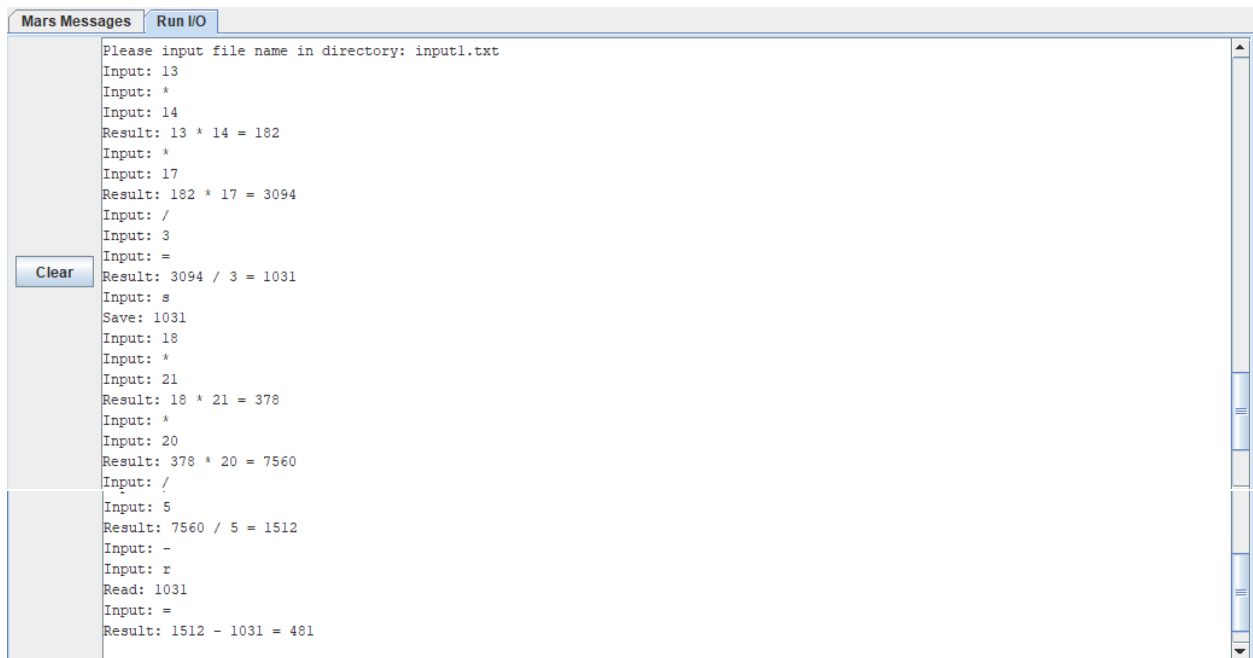


Figure 1 Input and Output Run Screen for file **input1.txt**

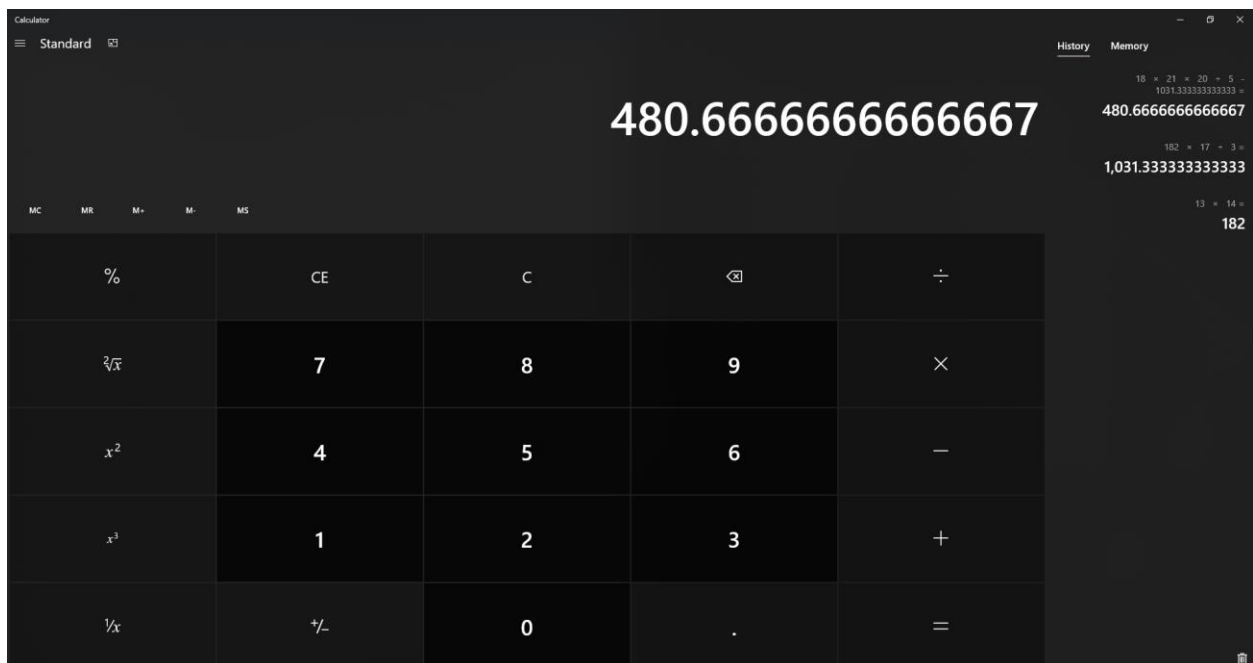


Figure 2 Screenshot from Microsoft Calculator with same operations

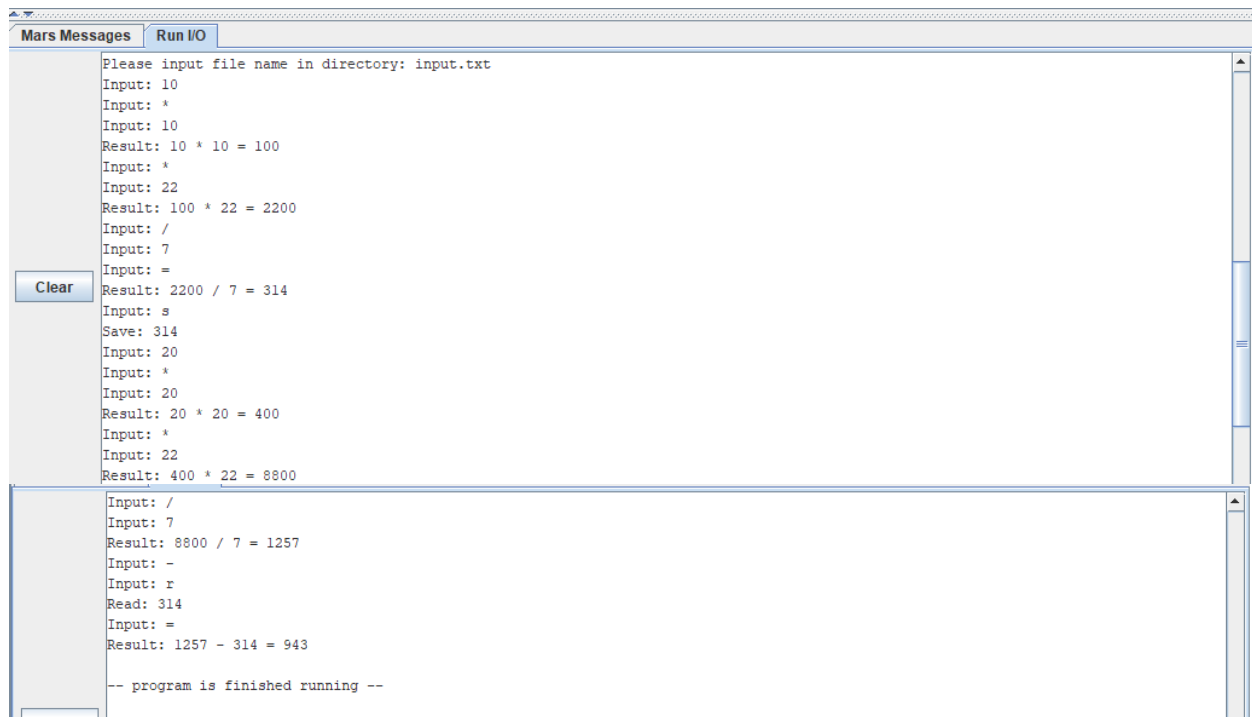


Figure 3 Input Output screen for file *input.txt*

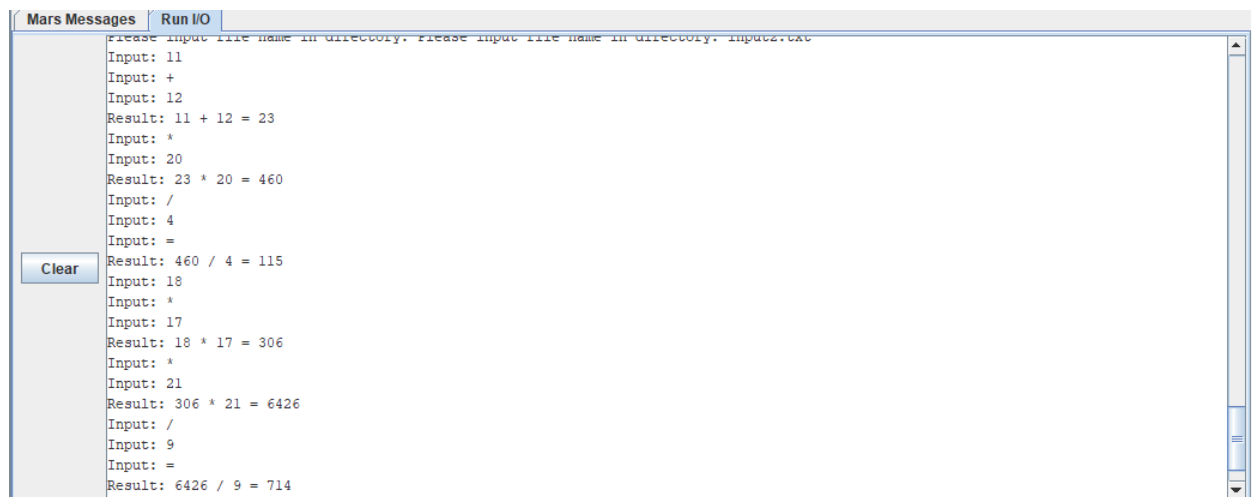


Figure 4 Input and Output Run Screen for file *input2.txt*

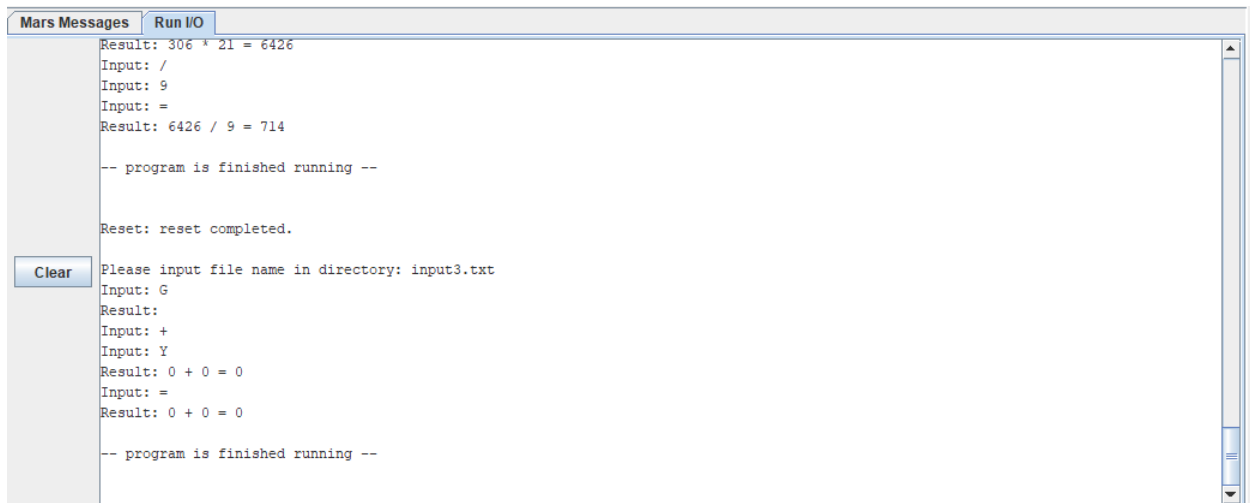


Figure 5 Input and Output Run Screen for file **input3.txt**

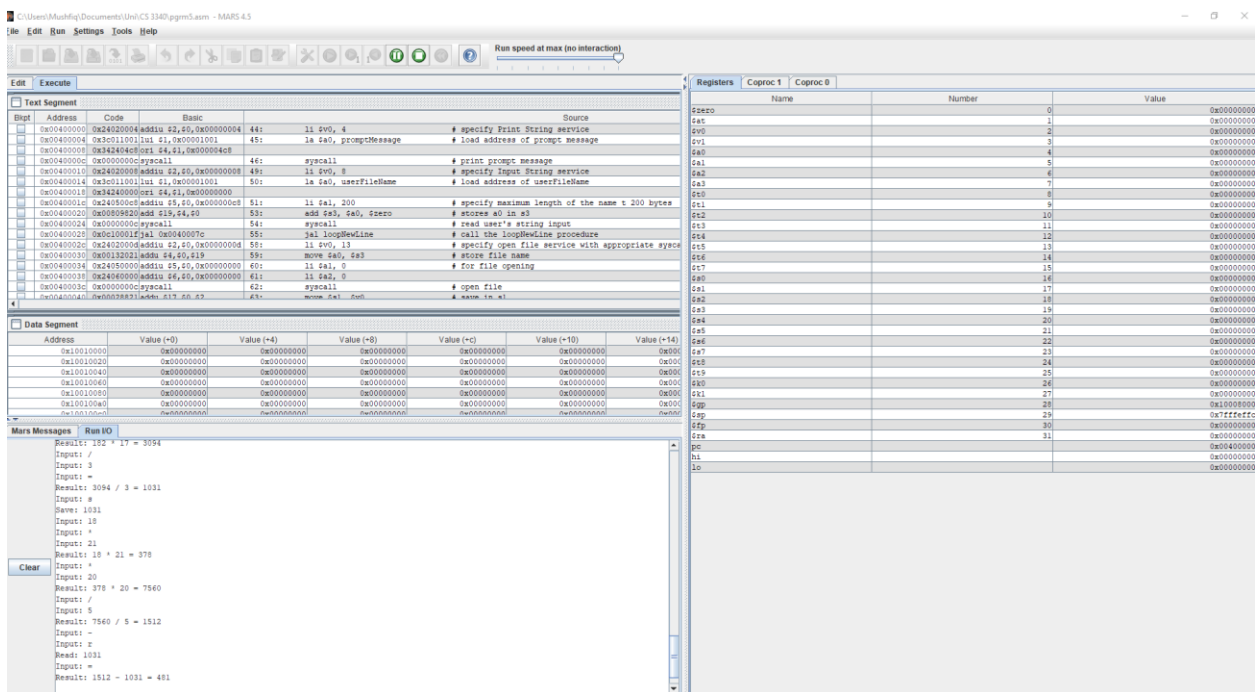
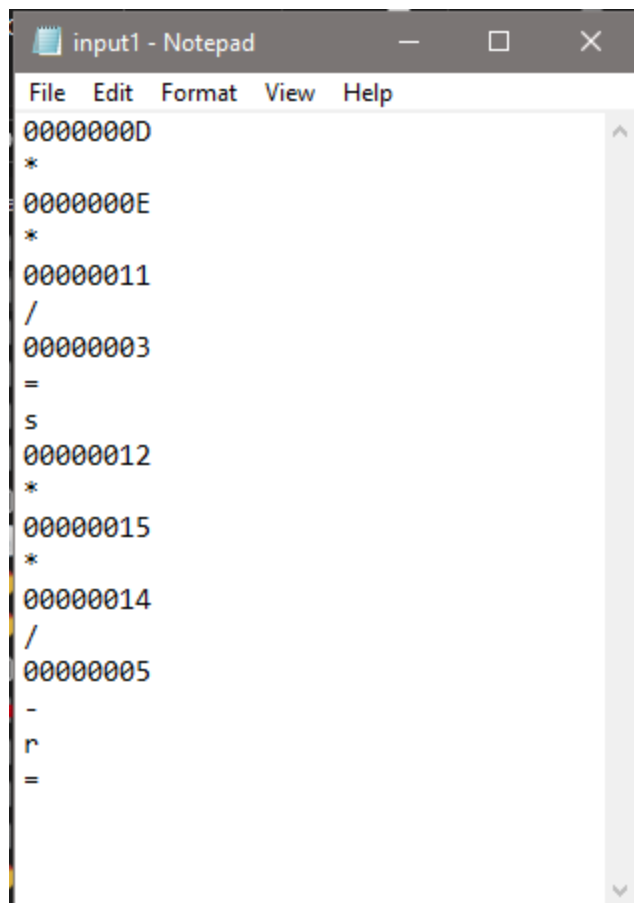
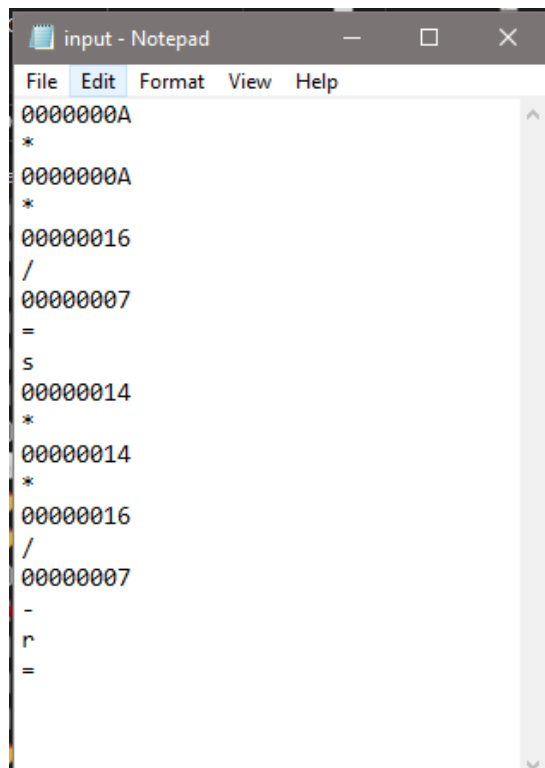


Figure 6 Execution screen after compilation



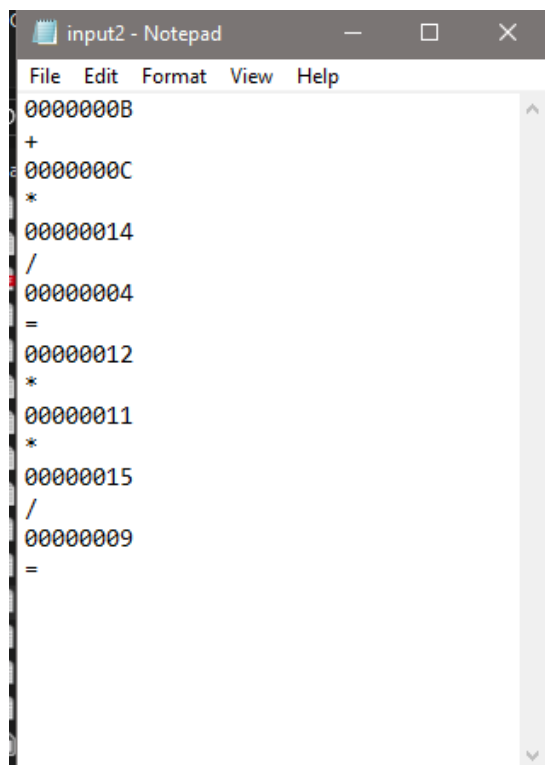
```
input1 - Notepad
File Edit Format View Help
0000000D
*
0000000E
*
00000011
/
00000003
=
S
00000012
*
00000015
*
00000014
/
00000005
-
r
=
```

Figure 7 input1.txt file



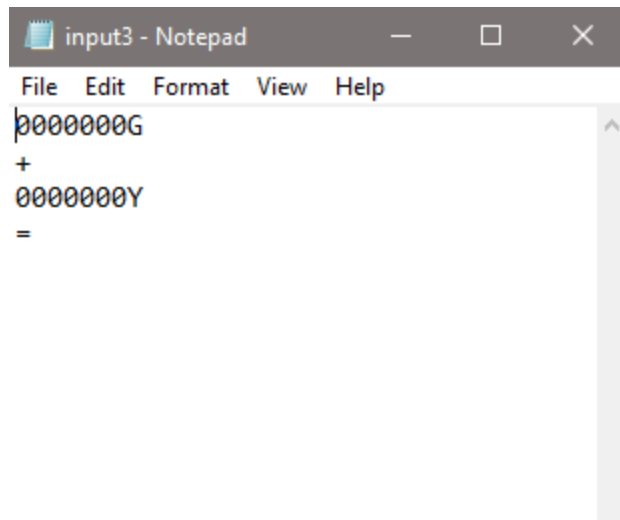
```
File Edit Format View Help
0000000A
*
0000000A
*
00000016
/
00000007
=
5
00000014
*
00000014
*
00000016
/
00000007
-
r
=
```

Figure 8 input.txt file



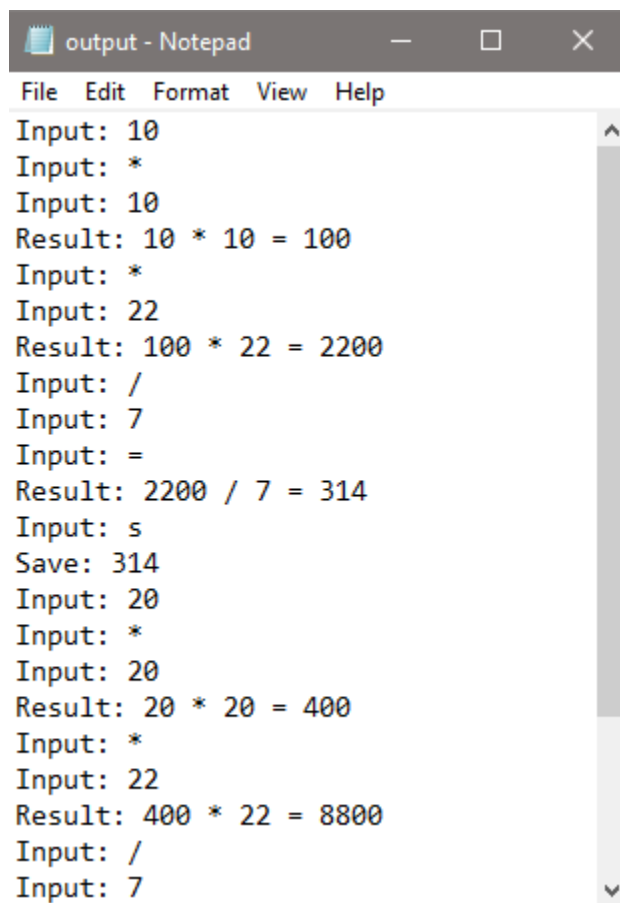
```
File Edit Format View Help
0000000B
+
0000000C
*
00000014
/
00000004
=
00000012
*
00000011
*
00000015
/
00000009
=
```

Figure 9 input2.txt file



```
input3 - Notepad
File Edit Format View Help
0000000G
+
0000000Y
=
```

Figure 10 input3.txt file



```
output - Notepad
File Edit Format View Help
Input: 10
Input: *
Input: 10
Result: 10 * 10 = 100
Input: *
Input: 22
Result: 100 * 22 = 2200
Input: /
Input: 7
Input: =
Result: 2200 / 7 = 314
Input: s
Save: 314
Input: 20
Input: *
Input: 20
Result: 20 * 20 = 400
Input: *
Input: 22
Result: 400 * 22 = 8800
Input: /
Input: 7
```

Figure 11 text file containing sample output for input.txt