COM S/SE 319 : Software Construction and User Interfaces
Spring 2018

Assignment 2

[Total Points: 100]

Assignment Due: Tuesday, April 24, 2018, 11:59 PM

[N.B.:**No Late submission will be considered for this Individual HW**]

This assignment is focused on Part 1: UI and event driven programming, and Part 2: Basic theory of grammars and parsing: Developing a Lexical Analyzer (Lexer) and a Parser.

Task 1: UI and Event Driven Programming: (50 points)

Objectives:

Learn to use Javascript objects, functions, and closures to implement UI and event driven programming.

Warm-up:

NOTE 1: One suggestion (to help you play with javascript) is to use online Javascript code tool like http://codepen.io/pen/ or https://jsbin.com. They are very useful for trying javascript examples as you can change the html or javascript directly on the website, and you can immediately see the results of your changes.

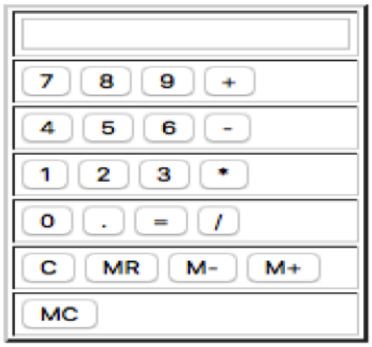
NOTE 2: You will need to also learn how to use the available tools for JS debugging.

Firefox has tools->WebDeveloper->Debugger,

Chrome has Tools->Developer Tools (ctrl-shift-I).

Task:

A complete example of another program (Matching game) is provided in **SampleProgram.zip**. Please take a look at that one first. A starting template is provided in **ExerciseHelp.zip**. Your assignment is to use this template to **create two simple calculator programs** (**one decimal and one binary**) using objects, functions, and closures. These calculators should look approximately like the below picture.



Look at a normal calculator to figure out the functionality of M+, M-, MR and MC.

For **binary calculator**,

1. Note that for some operations on the binary calculator, it may be more convenient to convert the binary numbers to integers and then do the operation.
2. You can assume that only positive binary numbers are represented and used. For example, positive 9 is represented as 1001.
3. Binary operator "+" represents plus operation (**2 points**)
4. Binary operator "*" represents multiply (**2 points**)
5. Binary operator "/" represents division (**2 points**)
6. Binary operator "%" represents mod or remainder (i.e. divide the first value by the second, what is remaining, only works on positive numbers) (**2 points**)
7. Unary operator "<<" represents one bit-shift left (i.e. insert zeros in the vacated position on the left, only works on positive numbers) e.g. (101 << gives 1010) (**2 points**)
8. Unary operator ">>" represents one bit-Shift right (insert zeros in the vacated position on the right, only works on positive numbers) e.g. (101 >> gives 10) (**2 points**)
9. Binary operator "&" represents AND (only works on positive numbers) e.g. (101 & 1011 gives 0001) (**2 points**)
10. Binary operator "|" represents OR (only works on positive numbers) e.g. (101 | 1010 gives 1111) (**2 points**)
11. Unary operator "~" represents not (i.e. invert each bit of the binary value, only works on positive numbers) e.g. (101 ~ gives 10) (**2 points**)

For **decimal calculator**,

1. " **+** ", " **-** ", " ***** " , " **/** " should be used respectively for addition, subtraction, multiplication and division. (**2x4 = 8 points**)
2. " **.** " should be used for operation with decimals. (**3 points**)
3. Negative number operations e.g., "(-2)-3 = -5" (**3 points**)
4. Assume that the calculator does not need to calculate complex operations such as 5 + 5 * 5. Instead, expect users to press "=" operator after a basic operation. So, press 5 + 5 followed by =. At this point it should show 10. Then, press "*****" and then 5 followed by "=". At this point show 50. When an operator button is pressed, the operator button's font becomes red. In other words, assume that we are expecting user to enter only "operand1 ***operator*** operand2 = ". However, we can use the results of the previous operation as the first operand for the next operation.

Check list:

[ ] Your javascript files should be named "calculator.js" and "calculatorBinary.js".
[ ] Use relative path in all of your files.
[ ] Name your Objects based on their purpose. Do the same with your JavaScript functions.
[ ] Show UI Display for decimal and binary calculator correctly. (**2*2 = 4 points**)
[ ] MR (shows memory value on screen) (**3 points**)
[ ] MC (clears memory value) (**3 points**)
[ ] M+ (Whatever is on screen gets added to memory) (**2 points**)
[ ] M- (Whatever is on screen gets subtracted from memory) (**2 points**)
[ ] C (clears screen value, clear the last operation, press "=" will not repeat the last operation) (**2points**)
[ ] Use "=" to show results of an operation and highlight the last button (any digit/ operator) clicked (**2 points**)
[ ] Make sure that your variables are not global (so that if someone includes some other js files with same names for variables, then your code still works ok).

What to Submit:

Make sure your solutions work on Chrome as TAs will use it to grade the assignment.
Submit via Canvas a **compressed file (.zip)** containing the following:

- lab.html, calculator.js, and calculatorBinary.js for Task 1
- README file explaining how to compile and run your program.

Task 2: Basic theory of grammars and parsing: Developing a Lexical Analyzer (Lexer) and a Parser (50 points)

You are required to develop a Lexical Analyzer (Lexer) and a Parser in JavaScript using the below template. **Show the output in GUI i.e., the output of the parser "Abstract Syntax Tree (AST)" and lexer "list of tokens" should be shown/visualized on the browser and also for the input which can be obtained from user as in input field on browser.**
Implement the whole program using a html page via an input textbox and a button to complete. Show that output as string.

Task 2.1: Implement a Lexer (15 points)

A lexer turns the input string into a list of tokens. A token looks the following way in javascript:

```
{
  "type": Symbol("Operator"),
  "value: "-"
}
```

For example, `lex` will turn the following expression:

```
mul 6 sub 4 sum 7 3 4
```

To the following array:

```
["mul", "6", "sub", "4", "sum", "7", "3", "4"]
```

Implement your Lexer based on the above description.

Task 2.2: Implement a Parser (15 points)

A parser turns the list of tokens into an Abstract Syntax Tree (AST). Visually, the parsing is a process which turns the array:

```
const tokens = ["sub", "3", "sum", "2", "4", "5"];
```

to the following tree:

```
 sub
 / \
3  sum
    / | \
   2 4  5
```

Below is the grammar that is used to parse the input token

```
num := 0-9+
op := sum | sub | div | mul
expr := num | op expr+
```

This translated to plain English, means:

- `num` can be any sequence of the numbers between 0 and 9.
- `op` can be any of `sum`, `sub`, `div`, `mul`.
- `expr` can be either a number (i.e. `num`) or an operation followed by one or more `expr`s.

Notice: 'expr` has a recursive declaration.

Task 2.3: Implement a Evaluator (5 points)

The evaluator visit each node from the tree with pre-order traversal and perform either of the following:

-Return the corresponding value, if the node is number type.
-Perform the corresponding arithmetic operation, if it is an operation node.

Task 2.4 Implement a code generator (5 points)

A code generator can translate the input to another language.

Task 2.5 Implement an interpreter (5 points)

An interpreter an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.

Task 2.6 Implement a compiler (5 points)

A compiler is a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer.

Implement the program using a html page as the UI. It contains one text box for input, and a "compile" button. Once the "compile" button is pressed, the result should be shown in the console.

For example:

Input: mul 5 sub 2 sum 7 2 9
Output in console should be:

- -80
- (5 * (2 - (7 + 2 + 9)))

What to Submit:

For Task 2, Make sure your solutions work on Chrome as TAs will use it to grade the assignment. Submit via Canvas a **compressed file (.zip)** containing source codes (.js, html etc.) and a README file explaining how to compile and run your program. You need to use above template for js source file and you can use HTML or other tools for showing GUI output.