Homework 3

The written part (PART I) is due **October 9**, Thursday (in class)

The programming part (PART II) is due **October 11**, Saturday midnight

On campus students must turn in a hard copy for the written part in class, with a cover page that has the student's name on it, which allows the grader to write the grade on the next page. **Distant –learning students must submit the written part on the Blackboard by midnight, October 9th. (This is to ensure the submissions to be received by the teaching staff.)**

For the submission requirement for the programming part, see PART 2.

PART 1 (The written part)

1.1 (35 pts).

The following grammar defines C-style array indexing:

P→A
A→ A R
A → id
R→ "[" A "]"

(1) Please convert the grammar into an LL(1) equivalent and then build an LL(1) parsing table for it without conflicts.

(2) Show how to parsing stack changes when applying your parsing table to input id[id[id]][id]

1.2 (15 pts).

The following grammar defines C-style pointer dereferences. In class we showed that the grammar is LALR(1). It is however, not LL(1). Please convert it into an LL(1) equivalent and build an LL(1) parsing table for it without conflicts. NOTE: You may need to go beyond the straightforward method of left factoring discussed in the lecture. Use the given grammar as the definition of the language, and then rethink how to rewrite it in LL(1).

S→ L = R ;
S → R ;
L → id
L → * R
R → L

PART 2 (50 pts). As in HW2, we consider a very small form of Java script. Please see HW2 PART 2 for the specification of our language, whose example was given below.

```
<script type="text/JavaScript">
var two = 2 ; var ten = 10
var linebreak = "<br />"

document.write("two plus ten = ")
var result = two + ten
document.write(result)
document.write(linebreak)

result = ten * ten
document.write("ten * ten = ", result)
document.write(linebreak)

document.write("ten / two = ")
result = ten / two
document.write(result)
var ID
ID = result
document.write(linebreak)
document.write(ID)

</script>
```

Please augment the lex/yacc (or flex/bison) program you wrote by adding semantic actions such that the parser produced will work as an interpreter of the given program. Notice that the only thing visible as the result of running the script would be what is printed by document.write() statements. You will interpret such statements by calling printf() standard functions appropriately.

For simplicity, we will assume that no string contains
 as a substring, except for the string "
" by its own. When the interpreter sees “
” as a parameter in document.write statement, or a variable that has the string value “
”, the semantic action should print a newline, i.e. “\n” in the C statement. All other strings should be printed in the exact way as quoted.

In order to evaluate the parameters that have numerical values to be printed, you will need to use semantic actions to evaluate expressions. The values of variables should be retrieved from a simple symbol table and your previous semantic actions should have stored the values in that table when the corresponding assignment statements were interpreted.

To check for undefined operations such as a string appearing in add/subtract expressions etc, you need to do basic type checking. Abort the execution of the offensive statement that contains such undefined operations, and then continue to execute the next statement.

If you do not like your own parser built for hw2, please contact the TAs to get a better one as your starting point.

Submission instruction for PART 2

1) For programming assignment, no offline submission will be accepted.
2) Use the following command at CS lab machines, e.g. the XINU machines (i.e., xinu01.cs ~ xinu20.cs) to submit your homework.
turnin –c cs502 –p pa3 [your working directory]

(Your home directories are shared amongst all CS lab machines.)

3) You MUST provide 'Makefile' for every programming question.
a. For example, your 'Makefile' of this assignment may look like

```
hw3:y.tab.c lex.yy.c
    gcc y.tab.c lex.yy.c -o hw3 -lfl
y.tab.c : hw3.y
    bison -y -d -g --verbose hw3.y
lex.yy.c:hw3.l
    lex hw3.l
clean:
    rm -f lex.yy.c y.tab.c
```

4) Your program MUST compile and run without any error at XINU machines. Please make sure your Makefile and program runs properly at XINU machines.
5) Any special instruction (if needed) for running your program should be included in the file named 'README'. By default, TA assumes your program could be run like this:
> ./hw3 program_name
Where hw3 is your executable file after compilation, and "program_name" is the file name containing the input program.

You may find information on the following links posted on piazza to be useful, including the following:
http://ds9a.nl/lex-yacc/cvs/lex-yacc-howto.html