# Fall 2016 CSE Qualifying Exam
# CSCE 531, Compilers

1. **LR-Parsing**

   (a) Give definitions of FIRST($\alpha$) and FOLLOW($X$).

   (b) Consider the following augmented grammar $G$ with start symbol $S'$:

$$
\begin{array}{rcl}
S' & \rightarrow & S \\
S & \rightarrow & V \text{ `=' } E \\
S & \rightarrow & E \\
S & \rightarrow & V \\
V & \rightarrow & \textbf{id} \\
V & \rightarrow & \text{`*' } E \\
E & \rightarrow & V
\end{array}
$$

   For the grammar $G$ above generate six of the LR(1) sets of items along with transition information for a canonical LR(1) parser

   (c) Using the partial Sets-of-Items constructed in part (b) construct as much as you can of the action table. If an action requires shifting to a state you did not construct, say "s?" for the action.

   (d) Describe in detail how the LR Parsing Algorithm will proceed in general when the state is $s$, the next token is $t$ and the stack contents is $X_0, X_1, \ldots, X_{top-1}, X_{top}$.

2. **Syntax-Directed Definitions** Consider the following standard LR (bottom-up) grammar for arithmetic expressions with constants, addition, and multiplication ($S$ is the start symbol):

$$\begin{array}{lcl} S & ::= & E \\ E & ::= & E_1 + T \\ E & ::= & T \\ T & ::= & T_1 * F \\ T & ::= & F \\ F & ::= & c \\ F & ::= & (E) \end{array}$$

(Subscripts have been added to distinguish different occurrences of the same nonterminal in the same production.) Do *either but not both* of the two items below:

(a) (For 80% credit) Add semantic rules to the grammar above that, given an input expression, produce an equivalent expression with the minimum number of parentheses. So the rules in effect remove unnecessary parentheses.

Your resulting expression should be passed as a string attribute to *S.output*. Assume that the terminal $c$ has a *text* attribute that contains the string representing the constant. You may use '+' in your actions to denote string concatenation, and please surround string constants with double quotes.

You should assume that $+$ and $*$ are associative operators, and that the usual precedence rules apply ($*$ before $+$). Do not rearrange or alter the expression in any way other than the parentheses.

(b) (For 100% credit) Do the same as the last item, but with the grammar also having a production for subtraction:

$$E \quad ::= \quad E_1 - T$$

Subtraction has the same precedence as addition, and both operators associate from left to right. Subtraction is not associative.

Examples:

| **Input** | **Output** | **Comment** |
|---|---|---|
| $2 + (3 + 4)$ | $2 + 3 + 4$ | addition is associative |
| $(2 * 3) * 4$ | $2 * 3 * 4$ | multiplication is associative |
| $2 + (3 * 4)$ | $2 + 3 * 4$ | multiplication presides over addition |
| $(2 + 3) * 4$ | $(2 + 3) * 4$ | parentheses needed |
| $(2 - 3) + 4$ | $2 - 3 + 4$ | operators associate left to right anyway |
| $2 - (3 + 4)$ | $2 - (3 + 4)$ | parentheses needed |

3. **Control Flow and Liveness Analysis**
Consider the intermediate code below.

```
     0    sum = 0
     1    i = 0
L0:  2    j = 0
L1:  3    t1 = b
     4    t1 = t1 + i * w
     5    t1 = t1 + j * 8
     6    f = array [ t1 ]
     7    if (f > 0) goto L2
     8    goto L3
L2:  9    sum = sum + f
    10    goto L4
L3: 11    sum = sum - 1
L4: 12    j = j + 1
    13    if (j < 32) goto L1
    14    i = i + 1
L5: 15    if (i < 8) goto L0
L6: 16    return
L7: 17    goto L0
```

Assume that there are no entry points into the code from outside other than at the start.

(a) (20% credit) Decompose the code into basic blocks B1,B2, ..., giving a range of line numbers for each.

(b) (20% credit) Draw the control flow graph, and describe any unreachable code.

(c) (40% credit) Fill in an 18-row table listing which variables are live at which control points. Treat `array` as a single variable. Assume that `n` and `sum` are the only live variables immediately before line 16 (the only exit point). Your table should look like this:

| **Before line** | **Live variables** |
|---|---|
| 0 | ... |
| 1 | ... |
| 2 | ... |
| ... | ... |
| 17 | ... |

(d) (20% credit) Describe any simplifying tranformations that can be performed on the code (i.e., transformations that preserve the semantics but reduce (i) the complexity of an instruction, (ii) the number of instructions, (iii) the number of branches, or (iv) the number of variables).

# Algorithms

1. Find tight asymptotic bounds on any positive function $T(n)$ satisfying the following recurrence for all large enough $n$:

$$T(n) = T\left(\frac{2n}{7}\right) + T\left(\frac{3n}{7}\right) + T\left(\frac{6n}{7}\right) + n^2$$

You may assume that any implicit floors or ceilings in the arguments to $T$ are of no consequence.

2. A **palindrome** is a string that is unchanged by reversal. Specifically, a string $S[1, \ldots, n]$ is a palindrome if $S[i] = S[n - i + 1]$ for every $i \in \{1, \ldots, n\}$. Examples: 'hannah', 'smhtiroglalgorithms', and 'x' are palindromes, but 'turtles' is not a palindrome.

Any string can be expanded into a palindrome by inserting characters. The table below shows some examples, with the inserted letters shown as capitals.

| **Original String** | **Palindrome** |
|---:|---|
| abb | abbA |
| turtles | SELtRurtles |
| smorgasbord | DROBsAGROmorgasbord |
| algorithms | SMHTIROGLalgorithms |
| kayak | kayak |

**Describe**, in pseudocode, a $\Theta(n^2)$ time algorithm that takes as input a string of length $n$, and outputs the length of the shortest palindrome that can be formed by inserting characters into the input string. Your algorithm only needs to compute the *length* of the palindrome; it does not need to generate the palindrome itself. **Explain** why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct.

(Hint: Let $p[i, j]$ denote the length of the shortest palindrome that can be formed by inserting characters into $S[i, \ldots, j]$. How can you fill in the table $p$?)

3. Consider a binary tree data structure represented as a collection of $n$ nodes, in which each node $v$ has three attributes:

(i) An integer *key* $v$.key. The keys are all distinct.

(ii) A pointer $v$.left to the *left child*. If $v$ has no left child, then $v$.left = NIL.

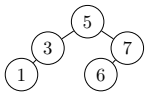
(iii) A pointer $v$.right to the *right child*. If $v$ has no right child, then $v$.right = NIL.

A binary tree is represented by a pointer to its root (or a NIL pointer, if empty).

This question has two parts; you should answer **both**.

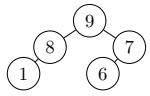
(a) (50% credit) **Describe**, in pseudocode, a $\Theta(n)$ time algorithm that decides whether a given tree is a *binary search tree*.

Hint: Be sure that your algorithm correctly identifies that this tree is a BST:




(b) (50% credit) **Describe**, in pseudocode, a $\Theta(n)$ time algorithm that decides whether a given tree is a *max-heap*. (Repeating for emphasis: The input is a binary tree represented by node objects with pointers to their children. The input is *not* the traditional array representation of a heap.)

Hint: Be sure that your algorithm correctly identifies that this tree is not a max-heap:




In both cases, the input is given as a pointer to the root node. **Explain** why your algorithms work, in enough detail to convince an intelligent but skeptical reader that they are correct.

# Architecture

1. For this question consider the following loop:

```
for (i=0;i<1024;i++)
  for (j=0;j<n;j++)
    a[j*128+i] = a[(j+2)*128+i] * 16;
```

a. Is this loop parallel? Why or why not?

b. Suppose you wanted to execute the inner loop in multiple threads. Describe a potential strategy for achieving this.

c. Is there a way to transform the loop nest to increase its potential parallelism? How or why not?

d. Using the original code, is it possible to unroll the inner loop without the compiler having access to the value of n? What are the potential advantages of unrolling the loop?

2. Consider the following segment of MIPS assembly code:

```
loop:   mov     r2,r3
        beq     r2,r8,complete
        addi    r4,r4,4
        lw      r3,0(r4)
loop2:  beq     r2,r3,exit
        sll     r5,r2,2
        lw      r6,0(r5)
        sw      r6,0(r7)
        addi    r7,r7,4
        addi    r2,r2,1
        j       loop2
exit:   j       loop
complete:
```

a. Identify all the data dependencies in the inner loop.

b. In what ways can a programmer parallelize this code?

c. Is there any instruction level parallelism available in this code?

3. Consider the following code:

```
float a[1024*1024], b[1024*1024], c[1024*1024];
for (i=0;i<1024;i++)
  for (j=0;j<1024;j++) {
    sum=0.0;
    for (k=0;k<1024;k++) sum+=a[i*1024+k]*b[k*1024+j]
    c[i*1024+j]=sum;
}
```

a. What is the arithmetic intensity of the i-loop, j-loop, and k-loop?

b. What is the minimum cache size needed to avoid any conflict and capacity misses in the k-loop?

c. How many times does the processor access the a-array before the same element is accessed twice? How many times does the processor access the b-array before the same element is accessed twice?