Fall 2019 CSE Qualifying Exam
CSCE 531, Compilers

1. **LR-Parsing.** Consider the following augmented grammar G with start symbol S':

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow + S\ S \\ S &\rightarrow * S\ S \\ S &\rightarrow - S \\ S &\rightarrow a \end{aligned}$$

   (a) For the grammar G above generate all of the LR(1) sets of items $I_0, I_1, I_2, \ldots$ along with complete transition information for an LALR parser.

   (b) Using the sets-of-items constructed in part (b), construct the action table and describe any conflicts. Assume the productions are numbered in order from 0 to 4.

   (c) Describe in detail how an arbitrary LR parsing algorithm will proceed in general when the next token is t and the stack contents are $s_0, s_1, \ldots, s_{top-1}, s_{top}$.

2. **A Syntax-Directed Definition.** Consider the grammar below for Boolean formulas (D is the start symbol):

$$\begin{array}{rcl} D & \to & D \vee C \\ & | & C \\ C & \to & C \wedge L \\ & | & L \\ L & \to & \neg L \\ & | & 0 \\ & | & 1 \\ & | & v \\ & | & (D) \end{array}$$

Disjunction ($\vee$) has lowest precedence, followed by conjunction ($\wedge$), then negation ($\neg$). Here v is a terminal that stands for a Boolean variable, and 0 and 1 stand for the Boolean constants FALSE and TRUE, respectively. Parentheses are used for grouping as usual.

Add semantic rules that compute, as an attribute of the parse tree's root, a simplified version of the input Boolean expression. You should handle the following simplifications (E is any Boolean expression):

$$\begin{array}{rr} E \vee 0 \mapsto E & 0 \vee E \mapsto E \\ E \vee 1 \mapsto 1 & 1 \vee E \mapsto 1 \\ E \wedge 0 \mapsto 0 & 0 \wedge E \mapsto 0 \\ E \wedge 1 \mapsto E & 1 \wedge E \mapsto E \\ \neg 0 \mapsto 1 & \neg 1 \mapsto 0 \\ (0) \mapsto 0 & (1) \mapsto 1 \end{array}$$

You need not perform any other simplifications. (The $\mapsto$ arrow just means that the left-hand side simplifies to the right-hand side.)

Note that simplifications should cascade if possible. For example,

$$\neg(v \vee 1) \;\mapsto\; \neg(1) \;\mapsto\; \neg 1 \;\mapsto\; 0$$

You can use the '+' operator for string concatenation.

3. Consider the intermediate code below.

```
 1       sum = 0
 2       i = 0
 3   L0: j = 0
 4   L1: t1 = b
 5       t2 = i * w
 6       t1 = t1 + t2
 7       t3 = j * 8
 8       t1 = t1 + t3
 9       f = a[t1]
10       t1 = 2 * t1
11       if f > 0 goto L3
12       goto L1
13   L2: sum = sum + f
14       goto L4
15   L3: sum = sum - 1
16   L4: j = j + 1
17       if j < 32 goto L2
18       i = i + 1
19       if i < 8 goto L0
20       goto L5
21       goto L0
22   L5: i = 0
23       no-op
```

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 23-row table listing which variables are live at which control points. Treat `a` as a single variable. Assume that `n` and `sum` are the only live variables immediately before line 23 (the only exit point). Your table should look like this:

Before line	**Live variables**
1	...
2	...
...	...
23	...

(d) (20% credit) Describe any simplifying transformations 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. **(Solving a Recurrence)** Let $T(n)$ be the function defined for all integers $n \geq 1$ by the following recurrence:

$$T(n) = n + \sum_{i=1}^{n-1} \frac{2T(i)}{3i}$$

Find an expression $f(n)$, as simple as possible, such that $T(n) = \Theta(f(n))$. Use the substitution method to **prove** that your answer is correct. [Hint: obviously, $T(n) = \Omega(n)$.]

2. **(A Pile of Books)** Suppose you have a collection of books of various sizes. You would like to construct the tallest pile you can from these books. However, to ensure that the pile is stable, you want to ensure that each book in the pile is strictly smaller in length, width, and height than the one below it.

Given a collection of n books $B_1, \ldots, B_n$, let ℓ_i, w_i, and h_i denote the length, width, and height of book B_i, respectively. We stay that B_i *stacks upon* B_j if $\ell_i < \ell_j$, and $w_i < w_j$, and $h_i < h_j$. An ordered sequence of books $B_{i_1}, \ldots, B_{i_m}$ is a *valid pile* if all of its books are unique and for all $1 \leq j < m$, B_{i_j} stacks upon $B_{i_{j+1}}$. The *height* of a valid pile is the total $h_{i_1} + \cdots + h_{i_m}$ of the heights of each of the books in the pile.

Describe an algorithm whose input consists of three arrays ℓ, w, and h, each containing n integers, describing the sizes of n books. The output should be the height of the tallest valid pile that can be formed from those books. Your algorithm only needs to compute the *height* of the tallest valid pile; it does not need to generate a list of books in this tallest valid pile. **Explain** why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct. For full credit, your algorithm should run in $\Theta(n^2)$ time.

3. **(Random Tree Nodes)** Recall the standard rotation-based balanced binary search tree data structures, such as AVL trees and red-black trees. These data structures store one key at each of their nodes, and have $\Theta(\log n)$ time operations to INSERT a new key (with appropriate rebalancing as needed) and to SEARCH the tree for a given key. For simplicity, assume that all of the keys in the tree are unique. To make this question concrete, you may consider AVL trees or red-black trees, or any other specific balanced binary search tree you prefer. (Hint: This choice of the rebalancing mechanism should probably not be relevant to your answer.)

Describe how to modify this data structure to also support a new GETRANDOMKEY operation, which should return a key chosen randomly from the keys stored in the tree. Each key in the tree should have equal probability to be selected. You may assume that a subroutine GETRANDOMINTEGER(ℓ, u) is available that can, in $\Theta(1)$ time, generate a random integer in the range $\{\ell, \ldots, u\}$.

Explain why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct. If your approach requires additional data to be stored at each node, define that information precisely, and explain how the standard operations should be modified to maintain that information. For full credit, each of INSERT, SEARCH, and GETRANDOM should run in $\Theta(\log n)$ time.

Fall 2019 CSE Qualifying Exam
CSCE 551, Theory

1. Fix a finite alphabet Σ. Given string $w \in \Sigma^*$, a *cyclic shift* of w is any string of the form yx where $x, y \in \Sigma^*$ are such that $w = xy$. Given language $L \subseteq \Sigma^*$, define

$$\text{cyclicShift}(L) \;:=\; \{yx \mid x, y \in \Sigma^* \wedge xy \in L\}\ ,$$

the language of all cyclic shifts of strings in L. Show that if L is regular, then cyclicShift(L) is regular. [Hint: Using an n-state NFA recognizing L, you can construct an NFA recognizing cyclicShift(L) with about $2n^2$ many states.]

2. Fix a finite alphabet Σ and let $L \subseteq \Sigma^*$ be a language such that

   (a) L contains at least one string of every length,

   (b) for any strings $x, y \in L$, either x is a prefix of y or y is a prefix of x,

   (c) there exists an infinite language $A \subseteq L$ such that A is Turing-recognizable.

   Show that L is decidable. (Recall that a string $x \in \Sigma^*$ is a *prefix* of a string $y \in \Sigma^*$ iff there exists a string $z \in \Sigma^*$ such that $y = xz$.) [Hint: How many strings of a given length can L have?]

3. Let EDGE HAMILTONIAN CIRCUIT (EHC) be the following decision problem:

   Input: A pair $\langle G, e\rangle$ where G is a graph and e is an edge of G.
   Question: Does G have a Hamiltonian circuit that traverses edge e?[1]

   Show that EHC is NP-complete. [Hint: for NP-hardness, try polynomially reducing to EHC from either HAMILTONIAN PATH or HAMILTONIAN CIRCUIT.]

[1] Equivalently, is there a Hamiltonian circuit that includes the endpoints of e consecutively?