CSci 365: Organizations of Programming Languages

Name: ______________________

Assignment 1: 70 + 10 points (optional)

Q1. [10] The switch statement in C language has the following format.

```
switch(expression) {
  case constant-expression:
          statement(s);
          break;          /* optional */

  /* you can have any number of case statements */

  default:                /* optional  */
  statements(s);
}
```

Write an Extended Backus Naur Form (EBNF) description for a **switch** statement, using the following non-terminals < ... >.

<switch_stmt> : specifies the switch statement abstraction
<expr> : specifies an expression,
<literal> : specifies a constant-expression,
<stmt_list> : specifies a list of statements.
switch, case, break, default: keywords in C.

Q2. [10] Using the following grammar in BNF, draw a parse tree and write a leftmost derivation for a statement below.

A = A / (B + (C / A))

Grammar in BNF:

<assign> → <id> = <expr>
<expr> → <expr> + <term> | <term>
<term> → <term> / <factor> | <factor>
<factor> → (<expr>) | <id>
<id> → A | B | C

Q3. [10] Rewrite the BNF of Q2 to give + precedence over / and force + to be right-associative.

Q4. [10] Modify the grammar of Q2 to add a unary minus (**-**) operator that has higher precedence than either + or /.

Note: A unary minus operator is, for example, **-** A in statement B = **-** A / 2.
Unary operators precede any operator.

Q5. [10]

(1) [5] Prove that the following grammar is ambiguous.

<S> → <A>
<A> → <A> - <A> | <id>
<id> → a | b | c

(2) [5] Modify (1) to be the unambiguous grammar in the simplest way.

Q6. [10] Compute the weakest precondition for the following assignment statements and postconditions:

a = 3 * (2 * b - a);

b = 2 * a – 1;

{b > 5} - postcondition

Q7. [10] Compute the weakest precondition for the following selection statement and postconditions:

if (x < y)

x = x + 1

else

x = 2 * x

{x < 0} - postcondition

Q8. [10, optional] Prove the following program is correct by applying Axiomatic Semantics:

{ $n > 0$ } - precondition

count = n;

sum = 0;

while count ≠ 0 **do**

sum = sum + count;

count = count - 1;

end

{sum = 1 + 2 + ... + n} - postcondition