CSE 6341, Written Assignment 3
Due Thursday, October 3, 11:59 pm (8 points)

Your submissions should be uploaded via Carmen. Create your answers using a text editor and upload the file (e.g., plain text, Word, PDF). Alternatively, you can write your answers by hand and take a photo (or scan), but please ensure that (1) your handwriting is *clear and legible*, and (2) your photo or scan has *high resolution*, to allow the grader to read and understand your submission.

Q1 (3 points): Consider the following context-free grammar for a C++ style language:
<program> ::= <classList>
<classList> ::= <classDecl> | <classDecl> **;** <classList>
<classDecl> ::= **class ident {** <classBody> **}** | **class ident**$_1$ **: ident**$_2$ **{** <classBody> **}**
<classBody> ::= … *[productions not relevant for this question]*

The program contains a sequence of class declarations. Each declaration contains the keyword **class**, followed by the name of the class. A class could optionally have a superclass: for example, declaration **class Y : X { … }** shows that class X is a superclass of class Y.

Define an attribute grammar to check the following condition: whenever a class declaration shows that some class Z is a superclass, that class Z must be declared earlier in the program. For example, the following program is valid
class X { … } ; class Y : X { …}
but this one is not
class Y : X { … } ; class X { …}

Your attribute grammar should accept all programs that satisfy this condition and should reject all programs that violate this condition. For each attribute you define, describe its name, type, and whether it is inherited or synthesized. Show the complete evaluation rules for all attributes. Assume a pre-defined attribute **ident**.*lexval*, as discussed in class.

*Try to keep your solution **as simple as possible**. Do **not** just blindly copy some solutions from the lecture notes. Do **not** use global data structures or side effects*.

Q2 (2 points): Consider the following context-free grammar, based on a grammar discussed in class:
<program> ::= <funcList>
<funcList> ::= <funcDef> | <funcDef> <funcList>
<funcDef> ::= **void ident ( ) {** <funcBody> **}**
<funcBody> ::= … *[productions not relevant for this question]*

Define an attribute grammar that accepts all and only programs in which *all function names are unique*—that is, no function name is used in multiple function definitions. For example, program

void f() { … } void g() { … }

should be accepted but program

void f() { … } void g() { … } void f() { … }

should be rejected.

Your solution *must use only **synthesized** attributes for non-terminals*. Your *solution must use **at most one** attribute per non-terminal*. Assume a pre-defined attribute **ident**.*lexval*.

Your attribute grammar should accept all programs that satisfy this condition and should reject all programs that violate this condition. For each attribute you define, describe its name, type, and whether it is inherited or synthesized. Show the complete evaluation rules for all attributes. *Try to keep your solution **as simple as possible**. Do **not** just blindly copy some solutions from the lecture notes. Do **not** use global data structures or side effects*.

Q3 (3 points): Consider the attribute grammar for assembly code generation discussed in class. Suppose we extend the language with a do-while loop, with the following syntax:

<stmt> ::= **do** <stmt>$_2$ **while (** <cond> **)**

Show the complete attribute grammar evaluation rules for generating assembly code for do-while loops. Use notation similar to the one used in the lecture notes.

Illustrate your solution by showing the complete generated code for the following program:

j=63; do { j=j*3; } while (j<999)

Use the code generation rules from the lecture notes together with your new rule. Since we have not defined code generation rules for conditional expressions such as **j<999**, just use **"..."** in your solution to denote the assembly code for computing the value of **j<999** and assume that this assembly code works as described in the lecture notes.

Synthesized Attributes
- **Definition**: The value of a synthesized attribute at a node is determined by the attribute values of its children.
- **Evaluation**: These attributes can be evaluated during a single bottom-up traversal of the parse tree.
- **Containment**: Both terminals and non-terminals can contain synthesized attributes.
- **Usage**: Synthesized attributes are used in both S-attributed and L-attributed syntax-directed translations (SDTs).

Inherited Attributes
- **Definition**: The value of an inherited attribute at a node is determined by the attribute values of its parent and/or siblings.
- **Evaluation**: These attributes can be evaluated during a single top-down or sideways traversal of the parse tree.
- **Containment**: Only non-terminals can contain inherited attributes.
- **Usage**: Inherited attributes are used only in L-attributed SDTs.

Example
Consider a production rule \(S \rightarrow ABC \):
- If \(S \)'s value is determined by the values of \(A \), \(B \), and \(C \), then \(S \) has a synthesized attribute.
- If \(A \)'s value is determined by the values of \(S \), \(B \), and \(C \), then \(A \) has an inherited attribute.