Instructions:
1. This is an open-book, open-notes exam.
2. All rules against cheating apply.
3. This exam begins at 630 pm and ends at 930 pm.
4. You will write your answers in softcopy, on your computer and upload the answer document to CSNS. MS Word, pdf, or text formats are acceptable. (If you need to submit a hardcopy pen-and-paper document, contact me for instructions.)
5. Be sure to put your name and student Id no. on your answer document.
6. The exam is graded out of a total of 60. The marks (points) for each question are indicated with the question.
7. You should have plenty of time; however you should manage your time wisely to avoid stress. As a guideline, the easier questions are towards the beginning and the harder questions are towards the end.

Bottom up parsing (19)

1. Consider the following grammar for describing a polygon:

   \( \text{Polygon } \rightarrow \text{ "POLY" PointList "END" } \)
   \( \text{PointList } \rightarrow \text{ Point } | \text{ Point PointList } \)
   \( \text{Point } \rightarrow \text{ "}num ", "num "} \)

   a. (3 points) Show a rightmost derivation of the polygon specification
      POLY (10,40) (50,90) END

   b. (1 point) Show the production that needs to be added to the polygon grammar to make it an augmented grammar.

   c. (4 points) Show the correct sequence of reducing productions in the bottom-up parse of the string
      POLY (10,40) (50,90) END

   d. (1 point) (True / False) The polygon grammar is left recursive and therefore cannot be handled by an LR(1) parsing technique.

   e. (1 point) (True/False) SLR parsing technique cannot parse regular grammars.

   f. (1 point) Give one example of an item for the polygon grammar.
g. (2 points) Construct I₁, the CLOSURE of Polygon \( \Rightarrow \text{POLY} \). PointList END

h. (3 points) Construct the GOTO(I₁, X) function value for I₁ for all possible values of X

i. (3 points) Show the values of : ACTION(I₁, X) for all possible values of X.

**Mid-term material (24)**

2. (2 points) Multiple choice: An interpreter differs from a compiler in the following way:
   a. Interpreter does not generate a separate module of executable code for the target machine, instead it directly executes the parsed source code
   b. Interpreter usually has fewer opportunities for optimization than a compiler
   c. There is no difference at all
   d. a and b
   e. Interpreter generates a runnable code module but it is for a virtual machine instead of the machine on which the compiler is running

3. (2 points) Multiple Choice: The term “relocatable machine code” implies
   a. The program counter is initialized with an absolute starting address at runtime.
   b. Assembly language code generated by the compiler always has absolute addresses in it.
   c. Compiler generates absolute addresses only for the variables in the heap area.
   d. a and c
   e. None of the above

4. (1 point) True/False: A context-free language is always context-sensitive.

5. (2 points) Multiple choice: The grammar
   \[ \text{thing} \Rightarrow \text{thing} + \text{thing} \mid \text{thing} \ast \text{thing} \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \]
   is
   a. context-free
   b. context-sensitive
   c. linear
   d. ambivalent
   e. a and b

6. (5 points) Rewrite the polygon grammar of Problem 1 in EBNF form.

7. 12 points) Using the MicroJava Expression grammar, show the top-down parse steps for the declaration
   \[
   \text{final int size} = 10;
   \]
Semantic Analysis, & symbol tables (9)

8. Consider the Variable Declaration part of a simplified MJ grammar, expressed as follows:

$$\text{VarDecl} \rightarrow \text{Type IdentList } ;;$$
$$\text{IdentList} \rightarrow \text{ident } | \text{ident }, \text{IdentList}$$
$$\text{Type} \rightarrow \text{ident}$$

You are required to parse this grammar segment and generate as output the grammar production that was used in the parsing step. For example, if your parser just processed the production

$$\text{IdentList} \rightarrow \text{ident }, \text{IdentList}$$

you should print out the string:

"IdentList ARROW ident, IdentList"

(and similarly for other productions)

a. (4 points) Augment the grammar above with attributes that are needed to generate the output as specified. Identify which attributes are synthesized and which are inherited.

b. (5 points) For the grammar above sketch the top-down parsing code along with semantic analysis code that builds on the attributes of part a above and correctly prints out the grammar production as specified. Is it necessary to use the symbol table for accomplishing this?

Code Generation (8)

9. (8 points) Consider the MJ code

```java
void main()
int[] a;
int b, c;
{
    a = new int[2]; // stmt 1
    a[1] = c + b;  // stmt 2
}
```

a (3 points) State the data areas of the microJava VM (covered in class) and sketch the allocation of the variables a, b, and c in their respective data areas.

b (2 points) Sketch the code area of the microJava VM as it appears at the end of the code generation step. (There is no need to show the detailed code for this part).

c. (3 points) Show the microJava VM code generated by stmt 2.