COP5621 Exam 3 - Spring 2005

Name: _____ (Please print)

Put the answers on these sheets. Use additional sheets when necessary. Show how you derived your answer when applicable (this is required for full cred it and helpful for partial credit). You can collect 100 points in total for this exam. A bonus question for an addit ional 15 points is included. If the total number of points for this exam exceed 100, the excess points are ca rried over to the next exams.

1. Match the terms below with the given sentences so as to best complete each sentence. Use no term more than once. Some terms will go unused. (10 points)

(1)	one	(8)	syntax-directed definitions
(2)	two	(9)	semantic actions
(3)	three	(10)	abstract syntax trees
(4)	four	(11)	translation schemes
(5)	attributes	(12)	parse trees
(6)	synthesized	(13)	inherited
(7)	S	(14)	L

   (a) There are two notations for associating semantic rules and actions with grammar productions: ___ and ___.

   (b) In translation schemes, ___ are inserted within the right-hand sides of productions.

   (c) Quadruples are an implementation of ___-address code statements.

   (d) The larger class of ___-attributed definitions enable translations to be performed without explicit construction of a parse tree.

   (e) The values of ___ attributes are passed upwards in a parse tree.

2. Give one reason why compilers generate intermediate code (rather than generating machine code directly). (5 points)

3. Given the following grammar for an expression:

$$
\begin{array}{lcl}
E & \rightarrow & E + T \\
E & \rightarrow & E \texttt{ - } T \\
E & \rightarrow & T \\
T & \rightarrow & T \texttt{ * } F \\
T & \rightarrow & T \texttt{ / } F \\
T & \rightarrow & F \\
F & \rightarrow & \texttt{(} E \texttt{)} \\
F & \rightarrow & \textbf{id}
\end{array}
$$

represent the string

a*(b-c)+(b-c)

as a *parse tree*, an *abstract syntax tree*, and a *DAG* that is minimal. (15 points)

4. Below is a grammar expressed in Yacc defining the syntax of a decimal constant. Add semantic actions to calculate the value of the constant and print the value to standard output. You may assume that Yacc's stack type (`YYSTYPE`) is `int`.(15 points)

```
decimal : digits
        ;
digits  : digits digit
        | digit
        ;
digit   : '0'
        | '1'
        ...
        | '9'
        ;
```

5. What are the three proposed *evaluation methods* for semantic rules to decorate parse trees? (5 points)

6. Consider the following syntax-directed definition:

$D \rightarrow T\ L$		$L.type := T.type$
$T \rightarrow$ **int**		$T.type := int$
$T \rightarrow$ **real**		$T.type := real$
$T \rightarrow$ **^** T_1		$T.type := pointer(T_1.type)$
$L \rightarrow V$		$V.type := L.type$
$L \rightarrow L_1$ **,** V		$L_1.type := L.type;\ V.type := L.type$
$V \rightarrow$ **id**		$enter($**id**$.entry,\ V.type)$
$V \rightarrow$ **id** [T]		$enter($**id**$.entry,\ array(V.type,\ T.type))$

(a) Identify the synthesized and inherited attributes. (5 points)

(b) Show the *annotated* parse tree for the sentence. (15 points)

^real id$_1$, id$_2$[int]

(c) Is the syntax-directed definition S-attributed? Is it L-attributed? (5 points)

7. Consider the following syntax-directed definition:

$$
\begin{array}{lll}
E \rightarrow E_1 + T & & E.val := E_1.val + T.val \\
E \rightarrow T & & E.val := T.val \\
T \rightarrow \mathbf{num} & & T.val := \mathbf{num}.val
\end{array}
$$

(a) Eliminate left recursion and show the converted translation scheme. (10 points)

(b) Write a recursive descent parser for the converted translation scheme. (5 points)

8. Suppose we build an interpreter in Yacc to evaluate integer and floating point expressions. The following fragment is part of this Yacc specification:

```
%union {
        Symbol  *sym;
        struct {
                int     type;
                union {
                        int    i;
                        float  f;
                } val;
        } rec;
}
%token <sym> ID
%type <rec> expr

%%

expr : expr '+' expr { ...



                     }
     | ID            { if ($1->type == Tint) {
                          $$.type = Tint;
                          $$.val.i = $1->val.i;
                       } else {
                          $$.type = Tfloat;
                          $$.val.f = $1->val.f;
                       }
                     }
     ;
```

The `expr` nonterminal has a `type` and a `val` field containing the type and value, respectively. Give the Yacc semantic actions to evaluate the sum '+' on integers and floats.(10 points)

9. **(bonus question).** (15 points) The following syntax-directed definition uses the backpatching technique to generate short-circuit code for Boolean expressions. The right-hand side of an assignment can be a Boolean expression. A 1 is to be stored in the assigned variable when the Boolean expression evaluates to true and a 0 is to be stored when the expression is false.

S	→	**if** E **then** M S	*backpatch*(E.t, M.q); *backpatch*(E.f, *nextquad*)
S	→	**id :=** E	*backpatch*(. . .) *emit*(**id**.place **:= 0**); *emit*(**goto** *nextquad*+1); *backpatch*(. . .) *emit*(**id**.place **:= 1**)
E	→	E_1 **or** M E_2	. . .
E	→	E_1 **and** M E_2	. . .
E	→	**(** E_1 **)**	. . .
E	→	**not** E_1	. . .
E	→	**id**	E.t := *makelist*(nextquad); *emit*(**if id**.place **goto** _); E.f := *makelist*(nextquad); *emit*(**goto** _)
M	→	ϵ	M.q := nextquad

(a) Complete the definition by completing the two backpatch operations in the assignment operation and the semantic rules for the **not** operator. (10 points)

(b) Generate intermediate code for

if not (a or b) then c := (a and b)

The first three-address statement is numbered 0 (i.e. *nextquad* returns 0 the first time). You may assume that **id**.place represents the identifier's name. Use additional sheets when necessary. (5 points)