**University of Toronto**
**Faculty of Arts and Science**
**April/May 2008 Examinations**
**CSC488H1S / CSC2107H1S**
**Duration – 2 hours ( 120 minutes )**
**OPEN BOOK ALL written aids, books and notes are allowed.**
**ALL non-programmable calculators allowed.**
**NO other electronic aids allowed.**

**120 marks total, 8 Questions on 4 Pages. ANSWER ALL QUESTIONS**
**Write all answers in the Exam book.**
**You must receive a mark of 35% or greater on this final exam to pass the course.**

**WRITE LEGIBLY** Unreadable answers cannot be marked.
Line/rule reference numbers on the left side of of programs and grammars are provided for ease of reference only and are not part of the program or grammar .
The notation **. . .** stands for correct code that has been omitted for brevity
State clearly any assumptions that you have to make to answer a question.

**1. [15 marks]** Assume that you wanted to improve the CSC488S project compiler by providing *source program* line numbers in *runtime* error messages. For example:

```
Execution Error - Divide by Zero on line 23 of the program.
```

Describe the *changes* in the project compiler

a) scanner
b) parser
c) AST data structure
d) code generation
e) run time environment
f) machine interpreter

that would be necessary to implement runtime line numbers.

**2. [15 marks]** In the lecture slides, a translation scheme for **switch** statements was given that assumed a compact set of case labels. Given a similar complete translations scheme (AST to IR mapping) for the case of vary sparse case labels ( e.g. 10, 1000, 10000, 1000000 ) where a branch table isn't practical. Include all details of expression handling and branching. You can invent any IR instructions you need but be sure to describe them clearly.

**3. [15 marks]** Consider the grammar G listed below

```
1       S  =  E ⊢
2       E  =  E '+' T ,
3             T
4       T  =  T '*' F ,
5             F
6       F  =  ident ,
7             '(' E ')'
```

Where `+ * ident ( )` are terminal symbols and ⊢ is end of file.

a) is this grammar LR( 0 )? Justify your answer.

b) is the grammar SLR( 1 ) ? Justify your answer.

c) is this grammar LALR( 1 )? Justify your answer.

d) convert this grammar into an LL(1) grammar that defines the same language.

**4. [15 marks]** A *typed union* (a *variant record* in Pascal) is a union like structure where a separate variable specifies which version of the union is active. Consider the following Pascal example:

```
 1   type shapeKind = (square, rectangle, circle);
 2   type shape =
 3       record
 4           centerx : integer;
 5           centery : integer;
 6           case kind : shapeKind of
 7               square : (side : integer);
 8               rectangle : (length, height : integer);
 9               circle : (radius : integer);
10           end;
```

Where the typed union is on lines 6 .. 9 and the record field *kind* is the union tag.

a) describe in the general case how the presence of a tag union would affect the algorithms for laying out storage for records.

b) describe the run time checking that would be necessary to ensure that the union is used correctly. Would any constraints on the programming language make this easier to do?

**5. [10 marks]** Assume that the programming language used in the course project has been extended with a *case expression* of the form:

```
1       primary          =  'case' expression 'of' '(' caseList ')
2       caseList         =  caseExpressions caseDefault
3       caseExpressions  =  caseExpressions oneCase ,
4                           oneCase
5.      oneCase          =  integer ':' expression
6       caseDefault      =  default ':' expression ,
7                           /* empty */
```

Where the *expression* after **case** is used to select one of the alternatives from the list of *caseExpressions*. If the *expression* does not match any of the *oneCase* alternatives then if a *caseDefault* is present, that is the value of the expression. If the *expression* does not match any of the *oneCase* alternatives and there is no *caseDefault* then the value of the expression is zero.
a) Describe the semantic analysis checking that should be performed on this statement to ensure that it is correct and reasonable. Show how the semantic checks will be implemented in the grammar (like the course project semantic analysis handout).

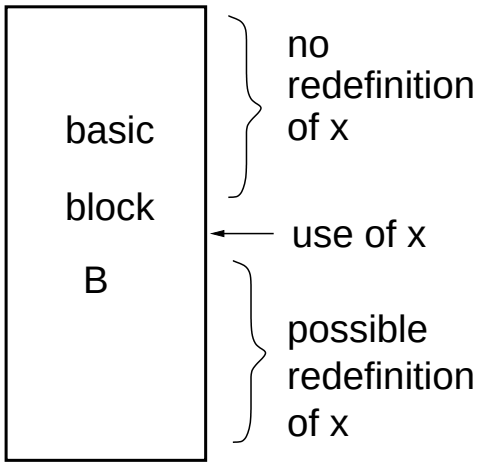
**6.[15 marks]** Design a code generation template for the case expression in the previous question, i.e. show how the general form of this expression should be transformed into machine instructions for the pseudo-machine used in the course project.

Show the code you would generate for the code fragment shown below

```
 1    bool isPrime
 2    int i
      . . .
12    isPrime =
13        case i of (
14            2 : false
15            4 : false
16            1 : true
17            3 : true
18            6 : false
19            5 : true
20            default : false
21        )
```

**7. [15 marks]** A data flow problem called *Upward Exposed Uses*. is used to detect use of a variable in a basic block before it is defined. This analysis could be used to detect variables that may be used before they are given a value.
To determine upward exposed uses we need a data flow algorithm that will compute for each definition the list of uses that it reaches (this is called a du-chain).

a) What is the direction of the analysis?
b) For a basic block B, what is the appropriate definition for $out(B)$ ?
c) How is the $use(B)$ set defined ?
d) How is the $def(B)$ set defined ?
e) What is the basic data flow equation for computing Upward Exposed Uses?



**8. [20 marks]** Show the optimizations that a good classic optimizing compiler will perform on the fragment of Java code shown below. You may show only the final result of the optimization if you make it very clear which optimizations have been performed.

```
 1    import Matrix ;
 2    public static Matrix magicSquare (int n) {
 3        double[][] M = new double[n][n];
 4        int P = n/2;
 5        int K = (n-2)/4;
 6        Matrix A = magicSquare(P);
 7        for (int J = 0; J  < P; J++) {
 8            for (int I = 0; I  < P; I++) {
 9                M[ I ][ J ] = A.get(I,J);
10                M[ I ][ J + P ] = A.get(I,J) + 2*P*P;
11                M[ I + P ][ J ] = A.get(I,J) + 3*P*P;
12                M[ I + P ][ J + P ] = A.get(I,J) + P*P;
13            }
14        }
15        for (int I = 0; I  < P; I++) {
16            for (int J = 0; J  < K; J++) {
17                double t = M[ I ][ J ];
18                M[ I ][ J ] = M[ I + P ][ J ];
19                M[ I + P ][ J ] = t;
20            }
21            for (int J = n - K + 1; J  < n; J++) {
22                double t = M[ I ][ J ];
23                M[ I ][ J ] = M[ I + P ][ J ];
24                M[ I + P ][ J ] = t;
25            }
26        }
          . . .
34        return new Matrix(M);
35    }
```