No: XXXX

CITY UNIVERSITY LONDON

Course name will be inserted here

IN2009 Language Processors

May 2009 **Time of exam**

Please read the instructions carefully

Answer ANY TWO questions.

All questions are worth 100 marks.

Calculators are allowed, except programmable ones

External Examiner: XXXXX Examiner: Dr. I. Siveroni

1)

a)

i) Explain what it means for a context-free grammar to be *ambiguous*.

[5 marks]

Answer: A grammar is ambiguous if two parse trees can represent two different derivations.

ii) From your explanation from i) above, show that the grammar below is ambiguous using the shortest string that will illustrate the ambiguity:

L	→	L O L
L	→	(L)
L	→	*true*
L	→	*false*
O	→	*or*
O	→	*and*

Answer: A demonstration of the two parse trees for the sentence true and false or true (more than one tree applies) for the full 15 marks, other indications of ambiguity are worth 5/10 marks depending on brevity.

[15 marks]

b) The reference manual for a MiniJava-like programming language contains the following grammar for a cut-down version of the *try* statement:

Statement → ***for*** *(id = Exp; Exp; id = Exp) Statement*

i) Sketch a possible abstract syntax for the *try* statement.

Answer: **`For Identifier Expression Expression Identifier Expression Statement`**

(be flexible – the answer may be expressed in Java with types, etc.)

[10 marks]

ii) Show how semantic actions in a grammar for a parser-generator such as JavaCC can be used to produce abstract syntax trees for the *try* statement.

Answer:

```
Statement ForStatement() :
{
  Statement s;
  Expression e1,e2,e;
  Id i1,i2;
}
{
  "for" "(" i1=Identifier() "=" e1=Expression()";"
   e=Expression()";"
   i2=Identifier() "=" e1=Expression()
   ")" s=Statement()
  { return new ForStm(i1,e1,e,i2,e2,s); }
}
```

(This might be expressed with symbols such as LBRACE instead of "{" etc.)
10 marks for frameworks & variables, 10 marks for syntax, 10 marks for correct actions.

[30 marks]

iii) Informally describe an appropriate typecheck for the *for* statement.

Answer: e must be a boolean expression. The type of e1 must match the type of i1, and the type of e2 must match the type of i1.

[10 marks]

iv) Suppose a compiler for a MiniJava-like language that includes the *for* statement translates all statements and expressions into intermediate code (e.g. intermediate representation (IR) trees). Outline the intermediate code that might be generated in translation of the *do/while* statement.
You may wish to use a simple example to explain your translation, e.g.:

```
for(x = 2; x < 20; x = x + 2)
// body of for
  { sum = sum + x; prod prod * x; }
```

You can assume that the expression tree for any variable `v` is simply `TEMP v`. Do not show translations for the body of the example *for* statement (in braces in this example `{...}`).

General Answer:

```
SEQ(Trans(e1),
  SEQ(LABEL(Lcond),
    SEQ(CJUMP(GT, TEMP x, CONST 20, Lbody, Lend)
      SEQ(LABEL(Lbody),
        SEQ(Trans(s),
          SEQ(Trans(e2),
            SEQ(JUMP(Lcond),
              LABEL(Lend)))))))
```

Trans(p) is the translation to IR of p. It might be expressed as trees.
15 marks for the comparison and jump, 5 marks for labelling loop start, 5 marks for labelling loop end, 5 marks for correct sequencing/ordering.

[30 marks]

[Total: 100 marks]

2)

a) The following regular expression recognises certain strings consisting of the letters a, b and c:

$$b^{+}\ [a]?\ (c|ab)^{*}$$

i) For the following 5 strings, indicate whether or not they are recognised by the above regular expression:

`accab, bbccca, bccabc, bbbba, baab`

Answer: no, no, yes, yes, yes.
All 5 strings must be listed and categorised – ones not listed and assumed to be invalid bear no marks.

[10 marks]

ii) Show three more strings that are recognised by the above expression.
Answer: Any three distinct strings that can be generated. 3 marks for one, 6 for two, all 10 marks for three. No extra marks for duplicates (either of the examples given, or of the candidate's own answer)

[10 marks]

iii) Show two more strings consisting of the letters a, b and c that are ***not*** recognised by the above regular expression.
Answer: Any two invalid strings not matched by the regExp. 2 marks for one, 5 marks for two. No extra marks for duplicates (either of the examples given, or of the candidate's own answer)

[5 marks]

b)

i) Explain why left-recursion must be eliminated from grammar productions which are to be used in construction of a recursive-descent parser.
Answer: Because the usual way of writing the procedures leads to immediate recursive call - 5 marks. Up to 3 marks if talking about infinite loops without details.

[5 marks]

ii) Write down a general rule for rewriting left-recursive grammar productions to equivalent right-recursive grammar productions.
Answer:
X → X a | b
(where a and b are strings of terms and non-terms) rewrites to:
X → b X′
X′ → a X′ | empty

[10 marks]

iii) Use the general rule from part ii) to rewrite the following productions to be right-recursive:

```
A → A and B
A → B
B → B <= C
B → C
C → ID
C → not ( A )
```

Answer (15 marks for rewritten production rules for A and B):
A → B A' *(5 marks)*
A' → and B A' | empty *(5 marks, 3 marks)*

B → C B' (5 marks)
B' → <= C B' | empty (5 marks, 3 marks)
C → ID | not(A) (2 marks, 2 marks)

[30 marks]

c) Consider the following Java class:

```
1  class Calculate {
2    String op; int total;
3    public void add(int rand, String msg) {
4      System.out.println(op);
5      int factor = 5;
6      int total = factor * rand;
7      System.out.println(msg);System.out.println(rand);
8      System.out.println(op);System.out.println(total);
9    }
10 }
```

Given an initial environment σ_0, derive the type binding environments for the method at each use of an identifier and indicate where type lookups will occur.

[30 marks]

Answer: (3 marks per entirely correct line, 2 marks if <u>mostly</u> correct, 1 mark if <u>partly</u> correct)

0. σ_0 = starting environment (={})
2. $\sigma_1 = \sigma_0$ + {op → String, total → int}
3. $\sigma_2 = \sigma_1$ + {rand → int, msg → String}
4. Lookup op in σ_1
5. $\sigma_3 = \sigma_2$ + {factor → int}
6. $\sigma_4 = \sigma_3$ + {total → int} overrides total in σ_1, lookup factor in σ_3 and rand in σ_2
7. Lookup msg in σ_2, rand in σ_2
8. Lookup op in σ_1 , total in σ_4
9. Discard σ_2, σ_3, σ_4, revert to σ_1
10. Discard σ_1, revert to σ_0

[Total: 100 marks]

3)

a) Why do many programming language implementations require a memory model that implements a runtime stack? Explain in detail how a stack frame is pushed to the stack, and removed from the stack, during program execution.
Answer: Procedure or method calls, recursion, need for separate storage space for parameters and locals. Code generated for a proc/func does the pushing/popping.

- ***`caller g(...) calls callee f(a1,...,an)`***
- ***`calling code in g puts arguments to f at end of g frame`***
- ***`stores return address`***
- ***`referenced through SP, incrementing SP`***
- ***`on entry to f, SP points to first argument g passes to f`***
- ***`old SP becomes current frame pointer FP`***
- ***`f then allocates frame by setting SP=(SP - framesize)`***
- ***`old SP becomes current frame pointer FP`***
- ***`f then initialises locals`***
- ***`on exit from f : SP = FP, removing frame`***
- ***`jumps to return address`***

10 marks for answer to first part and explanation. 20 marks for details.

[30 marks]

b) Some programming language implementations avoid in some circumstances the need to pass parameters via a stack frame. Outline what these circumstances might be and why passing via the stack frame might be avoided. Also, outline situations where the use of a stack frame to pass parameters cannot usually be avoided.
Answer: Appropriate when leaf procs, interproc reg alloc, dead variables, reg windows (but. . .).
Reg saves: when address is taken, when call-by-ref, when accessed by inner nesting, value too big, an array, convention of save for partic reg prior to call, spilling in exp evaluation, saving a reg window.
5 marks for explanation, 2 each for details.

[25 marks]

c) Suppose that a compiler translates a MiniJava-like language to an intermediate representation (for example IR trees) that will include the calculations required to address variables in stack frames. Draw or write down the intermediate representation required to access a local variable declared in a method. Explain your answer.
Answer: MEM(BINOP(PLUS,TEMP fp, CONST k)) where k is offset of var in frame, fp the register holding the framepointer. Has to compute place in frame.

[20 marks]

d) Explain the difference between *caller-save* and *callee-save* registers. Study the following methods and suggest for each whether a caller-save or callee-save register is appropriate for variable y. Explain your answers.

```
        int f (int x) { int y; y=x+1; g(y+1); return x; }

        int g (int x) { int y; y=y*x; f(x); return y; }
```

Answer: A register is caller-save if the code for the caller of a func must save and restore the register value around a func call. A register is callee-save if code for a func (the callee) saves and restores the value of the register.

- ***In method f, y should be in caller-save, since y does not live after y=x+1 (so code generated shouldn't save it).***
- ***In method g, y should be in callee-save, since y lives across the method calls.***

5 marks for explanation, 10 each for y answers and explanations.

[25 marks]