TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2008

MID-YEAR

COMP431

COMPILERS

Time Allowed: 3 Hours

Instructions:

- *Read each question carefully before attempting it.*
- This examination will be marked out of **180** marks.
- Answer all questions.
- You may answer the questions in any order. Make sure you clearly identify the question you are answering.
- Non-electronic foreign language-English dictionaries are permitted.
- Reference material and *calculators* are NOT PERMITTED.
- Use of mobile phones, laptop computers, PDAs or other electronic devices is NOT PERMITTED.

Questions	**Marks**
1. Preliminaries	[10]
2. Parsing	[18]
3. Type Checking	[40]
4. Intermediate Language	[25]
5. Peephole Optimisation	[30]
6. Register Allocation	[20]
7. Dataflow Analysis	[25]
8. Static Single Assignmnet (SSA)	[12]

Intermediate Language

In some of the questions, the following language (which is similar to JKil) will be used:

c	integers
v, m, f, C	local variable, method, field and class names (respectively)
e, e_1, e_2	arithmetic expressions
aop	binary arithmetic operators (e.g. `+`,`-`,`*`,`/`)
bop	binary boolean operators (e.g. `!=`,`==`,`<`,`>`,`<=`,`>=`)
L_1, L_2	instruction labels
D_C, D_M	class and method declarations (respectively)
T	Java types
S	statements

$$D_C ::= \texttt{class}\ C_1\ \texttt{extends}\ C_2\ \{\ \overline{T\ f}\ \overline{D_M}\ \}$$

$$D_M ::= T\ m(\overline{T\ v})\ \{\ \overline{T\ v}\ \overline{S}\ \}$$

$$T ::= \texttt{int} \mid C$$

$$S ::= [L_1]\ v = e \mid [L_1]\ v.f = e \mid [L_1]\ v_1 = v_2.m(t_1, \ldots, t_n) \mid [L_1]\ \texttt{return}\ t \mid [L_1]\ \texttt{if}\ v\ bop\ t\ \texttt{goto}\ L_2 \mid [L_1]\ \texttt{goto}\ L_2$$

$$e ::= t_1\ aop\ t_2 \mid v.f \mid \texttt{new}\ C() \mid t$$

$$t ::= v \mid c \mid \texttt{null}$$

You may assume that local variables must be declared and defined before being used in a method. An example program in the *Intermediate Language* is:

```
class Test extends java.lang.Object {
     int aMethod(int j) {
          int i,Test p
          [0] i=0
          [1] if i>10 goto 5
          [2] i=i+1
          [3] j=j*2
          [4] goto 1
          [5] j = this.aMethod(j)
          [6] return j
     }
}
```

Assume the class hierarchy follows Java and `java.lang.Object` is the root. So, for example, `java.lang.Integer` `extends` `java.lang.Number`, etc.

Question 1. Preliminaries [10 marks]

(a) [2 marks] Describe what is generally meant by a *compiler* in Computer Science.

(b) [8 marks] List all the generic compiler stages (this list should have at least four stages) and give a one sentence description of what happens at each stage.

Question 2. Parsing [18 marks]

(a) [5 marks] Describe *abstract syntax trees (AST's)* and outline what they are used for in compilation.

(b) [3 marks] Describe the difference between a *scanner* and a *parser*.

(c) [5 marks] In Pascal, semicolons are used to *separate* statements and brackets to *group* statements. This is illustrated in the following examples:

```
( statement ; statement ; statement )
( statement ; ( statement ; statement ) ; statement )
```

Write an unambiguous grammar which describes the above structure (HINT: `statement` will be one of the terminals).

(d) [5 marks] In C, semicolons are used to *terminate* statements and braces to *group* statements:

```
{ expression; expression; expression; }
{ expression; { expression; expression; } expression; }
```

Write an unambiguous grammar that describes this structure.

Question 3. Type Checking [40 marks]

Consider the following type system describing a trivial subset of the Java language. Its syntax is as follows:

$\texttt{L ::= class C } \{\overline{\texttt{C f;}}\ \texttt{K}\} \qquad \texttt{K ::= C}(\overline{\texttt{C f}})\{\overline{\texttt{this.f = f;}}\} \qquad \texttt{e ::= x | new C}(\overline{\texttt{e}}) \texttt{ | e.f}$

`L` is a class declaration for a class named `C` and $\overline{\texttt{C f}}$ stands for a list of fields each having a type `C` and a name `f`. `K` describes a constructor that must initialise all fields in the class. There are no methods or casts in our language and only three possible expressions: variable (`x`), class instantiation ($\texttt{new C}(\overline{\texttt{e}})$) and field access (`e.f`).

We provide the following expression type checking rules (where the $fields(\texttt{C})$ lookup function provides a list of fields given class `C`):

$$\frac{\Gamma \vdash \texttt{K} = \texttt{C}(\overline{\texttt{C f}})\{\overline{\texttt{this.f = f;}}\}}{\Gamma \vdash \texttt{class C}\ \{\overline{\texttt{C f;}}\ \texttt{K}\}\ \texttt{OK}}\ \text{(T-Class)} \qquad \Gamma \vdash \texttt{x} : \Gamma(\texttt{x})\ \text{(T-Var)}$$

$$\frac{fields(\texttt{C}) = \overline{\texttt{C f}} \qquad \Gamma \vdash \overline{\texttt{e : C}}}{\Gamma \vdash \texttt{new C}(\overline{\texttt{e}})}\ \text{(T-New)} \qquad \frac{\Gamma \vdash \texttt{e}_0 \texttt{ : C}_0 \qquad fields(\texttt{C}_0) = \overline{\texttt{C f}}}{\Gamma \vdash \texttt{e}_0\texttt{.f}_i \texttt{ : C}_i}\ \text{(T-Field)}$$

A program in our language is a list of class declarations followed by an expression. Each class declaration should meet the T-Class rule and the expression should meet the expression type rules. Assume that class `Object` is pre-defined. Also assume that a class table CT maps class names `C` to class declarations `L` and an environment Γ maps variables to their types.

(a) [10 marks] Construct an example of a program in this language that contains at least one class declaration with at least two fields and an expression that uses each of the valid kinds of expression.

(Question 3 continued on next page)

(Question 3 continued)

(b) For each of the following programs show the program derivation if it type checks or show why it does not type check.

(i) [8 marks]

```
class A {
  Object f;
  A(Object f) { this.f = f; }
}

class B {
  A fA;
  Object fO;
  B(A fA, Object fO) { this.fA = fA; this.fO = fO; }
}

(new B(new A(new Object()), new Object())).fA.f
```

(ii) [8 marks]

```
class Pair {
  Object fst;
  Object snd;
  Pair(Object fst, Object snd) { this.fst = fst; this.snd = snd; }
}

(new Pair(new Object(), new Pair(new Object(), new Object()))).snd.fst
```

(c) [4 marks] In your own words, briefly discuss the purpose of a *type system*. You may choose to refer to the rules and examples given in this question.

(d) [5 marks] Briefly describe the purpose of subject reduction and progress proofs. State what they show for the language type system described in this question.

(e) [5 marks] In the description of the language we have presented rules for syntax, type rules, and a $fields$ lookup function. Explain what other category of rules must also be provided to fully describe a language behavour using a type system?

Question 4. Intermediate Language [25 marks]

(a) [4 marks] Discuss the differences between the Abstract Syntax Tree (AST) and Intermediate Language (or Representation) used in a compiler.

(b) Translate the following three Java programs into the *Intermediate Language* given on Page 2. You should attempt to produce intermediate language programs that are as efficient as possible.

(i) [7 marks]

```
class Foo {
 int f() {
  Foo foo = new Foo();
  if (foo != null)
   foo.f();
 }
}
```

(ii) [7 marks]

```
class Foo {
 int f(int a, int b) {
  if (a <= b && a > 0) {
   while (b > a) {
    b -= a;
   }
   return b;
  } else {
   return b;
  }
 }
}
```

(iii) [7 marks]

```
class Foo {
 int f(int y) {
  int z=y;
  if(z > 42) { y = 1 + (z*2); }
  else { y = (2*z) + 2; }
  return z;
 }
}
```

Question 5. Peephole Optimisation [30 marks]

(a) [5 marks] Describe and discuss the peephole optimisation technique.

Consider the following peephole optimisation rules:

	Original Form	Optimised Form
1	`..iload X; iconst Y; iadd; istore X;..`	⟶ `..iinc X,Y;..`
2	`..iconst Y; iadd; istore X;..`	⟶ `..istore X; iinc X,Y;..`
3	`..bipush X; bipush X;..`	⟶ `..bipush X; dup;..`
4	`..Tconst X; Tconst Y; Tadd; ..`	⟶ `..Tconst X+Y;..`
5	`..aconst_null;aload_X; if_acmpne L1;..`	⟶ `..aload_X; ifnonnull L1..`
6	`..if_icmpge L1;...; L1: goto L2;..`	⟶ `..if_icmpge L2;...; L1: goto L2;..`

(b) [5 marks] Give at least 2 more rules not listed above.

(c) [10 marks] Consider the following bytecodes:

```
      iconst_1
      istore_1
      iload_1
      iconst_1
      iadd
      istore_1
      bipush    10
      newarray  int
      astore_2
      iconst_0
      istore_3
L0:   iload_3
      bipush    10
      if_icmpge L2
      bipush    11
      bipush    11
      istore_1
      iconst_5
      istore    4
      iload     4
      goto      L1
L1:   iconst_5
      iadd
      istore    4
      iinc      3, 1
      goto      L0
L2:   goto      L3
L3:   return
```

Provide an optimised version of this code using the rules 1 . . . 6 listed above. Be sure to indicate which rules you have used and in which places.

(Question 5 continued on next page)

(Question 5 continued)

(d) [5 marks] Consider the following bytecodes:

```
        iload     1
        ifeq      L1
        iload     1
        iconst    2
        iadd
  L1:   istore    1
```

It might appear that Rule 1 from the above list could be applied here. Why is it unsafe to do so?

(e) [5 marks] Consider the following rule:

Rule 0: ...;Tstore X; Tload X;... $\longrightarrow$ ...;...

Discuss why this rule is unsafe to use in general. Give an example to illustrate the problem.

Question 6. Register Allocation [20 marks]

(a) [5 marks] Describe how register allocation works.

(b) [5 marks] Describe the graph colouring approach to register allocation.

(c) [5 marks] Given the following code draw the interference graph:

```
a = 1;
b = 2;
c = a + 3;
d = b + 4;
e = c + a;
a = b + a;
```

(d) [5 marks] Describe linear scan approach to register allocation.

Question 7. Dataflow Analysis [25 marks]

A compiler for the intermediate language given on page 2 should identify situations where a reference variable is known to be *non-null*. For example, consider the following:

```
int f(String x)
    String y
[0] y = "Hello World"
[1] z = y.size()
[2] if x == null goto 4
[3] z = z + x.size()
[4] return z
```

Here, the compiler infers that x is *definitely non-null* before statement 3, but *may be null* before statements 0, 1, 2, and 4. Likewise, variable y is determined to be *definitely non-null* before statements 1 - 4, and *may be null* before statement 0.

(a) [20 marks] Design a simple dataflow analysis for determining whether a variable is *definitely non-null* before a given statement. Be sure to describe the following parts of the analysis: lattice of values (including $\top$ and $\bot$), transfer function, and the generic dataflow equations.

To help, a skeleton for the different cases of the transfer function is given below:

$f_T(X,\ [L_1]\ v = e) =$???
$f_T(X,\ [L_1]\ v.f = e) = X$
$f_T(X,\ [L_1]\ v_1 = v_2.m(t_1, ..., t_n)) =$???
$f_T(X,\ [L_1]\ \texttt{return}\ t) = \bot$
$f_T(X,\ [L_1]\ \texttt{if}\ v\ bop\ t\ \texttt{goto}\ L_2) =$???, if $v\ bop\ t$ is true
$f_T(X,\ [L_1]\ \texttt{if}\ v\ bop\ t\ \texttt{goto}\ L_2) =$???, if $v\ bop\ t$ is false
$f_T(X,\ [L_1]\ \texttt{goto}\ L_2) =$???

$isNonNull(X,\ v) = \texttt{true}, \texttt{if}\ v \in X$
$isNonNull(X,\ v) = \texttt{false}, \texttt{if}\ v \notin X$
$isNonNull(X,\ c) = \texttt{false}$
$isNonNull(X,\ \texttt{null}) =$???
$isNonNull(X,\ t_1\ aop\ t_2) =$???
$isNonNull(X,\ v.f) =$???
$isNonNull(X,\ \texttt{new}\ C()) = \texttt{true}$

You need to fill in the missing bits marked with ???. Note that there are two cases for the `if` statement — one for the *false* and one for the *true* branch.

(b) [5 marks] Discuss whether or not your dataflow analysis can safely determine whether a field is *definitely non-null*.

Question 8. Static Single Assignment (SSA) [12 marks]

(a) [2 marks] State the key requirement relating variables and definitions that is enforced by the SSA form.

(b) [5 marks] Draw the control-flow graph of the following program:

```
int f (int i, int j) {
  int x, y;
  switch (i) {
    case 0:  x=3;
    case 1:  x=1;
    case 2:  x=4;
    case 3:  x=1;
    case 4:  x=5;
    default: x=9;
  }
  switch (j) {
    case 0:  y=x+2;
    case 1:  y=x+7;
    case 2:  y=x+1;
    case 3:  y=x+8;
    case 4:  y=x+2;
    default: y=x+8;
  }
  return y;
}
```

(c) [5 marks] Starting from the CFG in the previous subquestion, convert the program to SSA form. Be sure to include ϕ-nodes where appropriate.

****