# 2004 A SEMESTER EXAMINATIONS


THE UNIVERSITY OF
WAIKATO
*Te Whare Wānanga o Waikato*

| | |
|---|---|
| DEPARTMENT | Computer Science |
| PAPER TITLE | Programming Languages |
| TIME ALLOWED | Three Hours |
| NUMBER OF QUESTIONS IN PAPER | Six |
| NUMBER OF QUESTIONS TO BE ANSWERED | Five |
| VALUE OF EACH QUESTION | All questions are of equal value. |
| GENERAL INSTRUCTIONS | Answer ANY FIVE questions. |
| SPECIAL INSTRUCTIONS | Nil |
| CALCULATORS PERMITTED | No |

1. (a) Given the following grammar and its First and Follow sets below, construct its LL(1) parsing table. The start symbol is Bexpr. Nonterminals start with an uppercase letter. Terminals are bolded in lowercase.

Bexpr → Bterm Bexpr'

Bexpr' → **or** Bterm Bexpr' | **ε**

Bterm → Bfactor Bterm'

Bterm' → **and** Bfactor Bterm' | **ε**

Bfactor → **not** Bfactor | **(** Bexpr **)** | **true** | **false**

| | | | |
|---|---|---|---|
| First(Bexpr) = { not, (, true, false } | | Follow(Bexpr) = | { $, ) } |
| First(Bexpr') = { or, **ε** } | | Follow(Bexpr') = | { $, ) } |
| First(Bterm) = { not, (, true, false } | | Follow(Bterm) = | { $, ), or } |
| First(Bterm') = { and, **ε** } | | Follow(Bterm') = | { $, ), or } |
| First(Bfactor) = { not, (, true, false } | | Follow(Bfactor) = | { $, ) or, and } |

[8 Marks]

(b) Outline the structure of a Lex (Flex) program, i.e. describe the various sections in order and describe what is placed in each section.

[6 Marks]

(c) Describe the lexical analysis and syntax analysis phases of a compiler.

[6 Marks]

2. (a) Discuss three differences between pure functional programming languages and imperative programming languages. In your discussion you should consider the significance of these differences in terms of issues such as programming style and program correctness.

[8 Marks]

(b) The following Haskell code removes the vowels from a string and encloses the string in brackets. Rewrite the function `processString` using the `map` function

```
removeVowels xs
   = [x | x <- xs, not (isaVowel x)]

bracket xs = "(" ++ xs ++ ")"

bracketedWithoutVowels xs
    = bracket (removeVowels xs)

processString  [] = []
processString  (x:xs)
  = bracketedWithoutVowels x ++ processString  xs
```

[3 Marks]

(Question 2 continued next page)

(c) What is the type of the function definition `what_type` below?

```
what_type = filter (>0). map (+1)
```

[2 marks]

(d) What is the result of the function call?

```
what_type [-3, 6, 7, 0, -1]
```

[2 Marks]

(e) Using the example code below discuss the notion of a polymorphic type as used in Haskell generic functions and how Haskell ensures they are type safe.

[5 Marks]

```
elem :: Eq a ⇒ a -> [a] -> Bool
elem x [] = False
elem x (y:ys)
    = (x == y) || elem x ys
```

3. (a) Write a paragraph describing the LL(1) parsing method.

[5 Marks]

(b) Describe the characteristics of each of the three type equivalence approaches, i.e. structural, declaration and name equivalence.

[5 Marks]

(c) The programming language D is a C-like language, which uses structural equivalence for determining the type equivalence of arrays and structures. Given the following type declarations, which structure declarations is structure A **NOT** equivalent to. Justify your answers.

[5 Marks]

```
typedef int anarray[10];
struct A {
          anarray x;
          int y;
         };
struct B {
          int x[10];
          int y;
          };
struct C {
          int y;
          int x[10];
          };
struct D {
          anarray a;
          int b;
          };
```

(Question 3 continued next page)

(d) Write a short paragraph discussing static versus dynamic type checking.

[5 Marks]

4. **Prolog**

(a) Suppose you are given tuples for a `move/2` relation as below. Use Prolog to implement a general test predicate `contains_loop()`, that should succeed if the directed graph described by `move/2` contains a cycle. The predicate should fail for acyclic graphs. The correct answer for the given example would be success, as this graph contains multiple cycles.

```
move(1,6).
move(1,8).
move(6,1).
move(6,7).
move(8,1).
move(8,3).
move(7,6).
?- contains_loop()  →  yes
```

[7 Marks]

(b) For the following six unification problems state whether they fail or succeed, and in case of success, list the resulting bindings for all variables involved:

1. p(a,b,b) = p(X,Y,Z)
2. q(Y,g(a,b)) = q(g(X,X),Y)
3. older(father(Y),Y) = older(father(X),john)
4. knows(father(Y),Y) = knows(X,X)
5. append([1,2,3|S1],S2,[1,2,3,4]) = append([X|L1],L2,[X|L3])
6. f(X,g(Y,h(Z,1),2),3) = f(1,g(2,h(3,U),V),W)

[6 Marks]

(c) Write a Prolog predicate `triplets(X,Y,Z)`, that computes via backtracking all solutions according to the following specification: X, Y, and Z are all digits from 0 to 9 (inclusive), X, Y, and Z are different from each other, and the following equation holds: (10*X + Y) / (10*Y + Z) = X / Z.

[7 Marks]

## 5. GC Algorithms

Select four of the GC algorithms listed below and (a) briefly describe how they work, as well as (b) discuss advantages and disadvantages of each of the four algorithms you have selected:

reference counting

mark-and-sweep

mark-and-compact

copying GC

generational GC

[20 Marks]

## 6. Java

(a) The following code example demonstrates the use of overloaded methods. What output does this program produce?

[8 Marks]

```
class A {
    public void f(Object o) {
System.out.println("object"); }
    public void f(A a) { System.out.println("an a");}
}
class B extends A {
    public void f(B b) { System.out.println("a b"); }
}
class Test {
    public static void main(String[] args) {
        Object o = new A();
        A a = (A) o;
        Object o1 = new B();
        B b = (B) o1;
        a.f(a);
        a.f(b);
        a.f(o);
        a.f(o1);
        b.f(a);
        b.f(b);
        b.f(o);
        b.f(o1);
    }
}
```

(b) List and explain four constraints a user-defined equals-method must obey. Additionally, if a class implements its own specialized equals-method, does it also have to supply an appropriate a) specialized hashCode-method and/or b) specialized compareTo-method.

[12 Marks]