Information and instructions:

- The questions are divided into three groups. The questions in the first group ask you to explain some basic concepts in programming language design. The questions in the second group ask you to demonstrate your understanding of the basic principles underlying certain programming language constructs. The questions in the third group are problem solving questions. Make sure you allocate sufficient time to the questions in this latter group.

- Provide your answer in the box after each question. If you absolutely need extra space, use the backs of the pages, but try to avoid it. The size of each box is an indication of the length of the answer I expect.

- You are not allowed to use a cheat sheet.

- Read every question carefully before answering.

- Do not forget to write your banner number and name on the top of this page.

- This exam has 13 pages, including this title page. Notify me immediately if your copy has fewer than 13 pages.

- The total number of marks in this exam is 100.
1 Basic Concepts

Question 1.1 (Different automata, different languages)  

(a) What types of languages can be accepted by a deterministic finite automaton?

(b) Are non-deterministic finite automata more powerful than deterministic finite automata, that is, is there a language that can be recognized by one but not by the other.

(c) Explain informally the difference between a finite automaton and a push-down automaton.

(d) What types of languages can be accepted by a push-down automaton?

(e) What types of languages can be accepted by a deterministic push-down automaton?
Define the following terms.

(a) Static (lexical) binding

(b) Dynamic binding

(c) Shallow binding

(d) Deep binding

(e) Applicative order evaluation

(f) Normal order evaluation
(a) What is a type system and why is it a good idea for a programming language to have a type system?

(b) What is name equivalence of types?

(c) What is structural equivalence of types?

(d) What is a statically typed language?

(e) What is a dynamically typed language?
Question 1.4 (Garbage collection) 7 marks

Describe a garbage collection algorithm of your choice. The algorithm must be able to reclaim useless objects also in the presence of circular structures.
2 What Does it Do?

Question 2.1 (Variable Binding) 12 marks

Consider the different combinations of variable binding and binding of function arguments and list
for each combination the output produced by the following Scheme-like program.
Note: display prints its argument, much like Java's println. newline prints a newline.

(define x 0)
(define y 0)
(define (f z) (display (+ z y)) (newline))
(define (g f) (let ((y 10)) (f x)))
(define (h) (let ((x 100)) (g f)))
(h)

<table>
<thead>
<tr>
<th>Variable binding</th>
<th>Function argument binding</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Dynamic</td>
<td>Shallow</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Question 2.2 (Short-circuit evaluation) 6 marks

Consider the following Java code.

```java
class ShortCircuit {
    public static void main(String[] args) {
        ShortCircuit a = new ShortCircuit(), b = null;
        int x = 3, y = 4;
        if(x != y || a.f()) System.out.println(b.f());
    }
    ShortCircuit() { z = null; }
    boolean f() { return z.g(); }
    boolean g() { return true; }
    ShortCircuit z;
}
```

Given that Java uses short-circuit evaluation of Boolean expressions, what is the output produced by
this program? Justify your answer.

Assuming Java did not use short-circuit evaluation of Boolean expressions, what would the output
be? Justify your answer.
Question 2.3 (Constructors and assignment) 12 marks

Here are two programs which should be assumed to compute the same thing. However, one of them is significantly faster than the other. How long does each program run, assuming that the time taken by anything whose running time is not noted in a comment is negligible? Justify your answer.

**Program 1**

class A {
    public:
        A() { /* 5 seconds */ }
        A(const A& x) { /* 6 seconds */ }
        const A& operator =(const A& x) { /* 6 seconds */ }
        const A& f() { return *this; }
    }

    int main() {
        A a;
        A b = a.f();
    }
}

**Program 2**

class A {
    public:
        A() { /* 5 seconds */ }
        A(const A& x) { /* 6 seconds */ }
        const A& operator =(const A& x) { /* 6 seconds */ }
        A f() { return *this; }
    }

    int main() {
        A a;
        A b;
        b = a.f();
    }
}
Consider the following C++ code.

class A {
    public:
        void f() { cout << "A::f "; }
        virtual void g() { cout << "A::g "; f(); }
    }

class B : public A {
    public:
        void f() { cout << "B::f "; }
        void g() { cout << "B::g "; f(); }
    }

int main() {
    A a;
    B b;
    A *x = &b;
    a.g(); cout << endl;
    b.f(); cout << endl;
    x->f(); cout << endl;
    x->g(); cout << endl;
}

What does this program output?
Question 3.1 (Finite automata) 5 marks

Provide a graphical representation of a DFA that recognizes the language of all binary strings with either an even number of 1s or a number of 0s divisible by three, but not both.
Provide a recursive-descent parser for the language described by the following context-free grammar.

<table>
<thead>
<tr>
<th>Rule</th>
<th>PREDICT Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Prog} \rightarrow \text{Stmt } ) $</td>
<td>{\text{if, while, begin}}</td>
</tr>
<tr>
<td>( \text{Stmt} \rightarrow \text{if } \text{Expr} \text{ then } \text{Stmt} \text{ else } \text{Stmt} )</td>
<td>{\text{if}}</td>
</tr>
<tr>
<td>( \text{Stmt} \rightarrow \text{while } \text{Expr} \text{ do } \text{Stmt} )</td>
<td>{\text{while}}</td>
</tr>
<tr>
<td>( \text{Stmt} \rightarrow \text{begin } \text{Stmts} \text{ end} )</td>
<td>{\text{begin}}</td>
</tr>
<tr>
<td>( \text{Stmts} \rightarrow \text{Stmt} \text{ ; } \text{Stmts} )</td>
<td>{\text{if, while, begin}}</td>
</tr>
<tr>
<td>( \text{Stmts} \rightarrow \varepsilon )</td>
<td>{\text{end}}</td>
</tr>
<tr>
<td>( \text{Expr} \rightarrow \text{id} )</td>
<td>{\text{id}}</td>
</tr>
</tbody>
</table>
Haskell allows the specification of infinite data structures. This does not cause any trouble, as long as only a finite portion of the data structure is ever used. For example, the code

```haskell
f :: Int -> Int
f x = x * x

putStrLn $ take 10 [f x | x <- [1..]]
```

prints `[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]`, even though `[f x | x <- [1..]]` is the infinite list of all squares of positive integers.

Provide a C++ or Java implementation that can be used to represent infinite lists of integers. In particular, provide three classes `List`, `Function`, and `Square`. `List` should provide the following methods:

- The static method `intList()` builds an infinite list containing the positive natural numbers `[1, 2, 3, ...]`.
- `head()` returns the first element of the list.
- `tail()` returns the tail of the list (the list without its head).
- `nth(int i)` returns the ith element of the list, where the head of the list has index 0.
- `map(Function f)` constructs a new List obtained by applying f to each element of the current list.

`Function` should provide one abstract method `apply(int x)`, which computes a new integer from x and returns it. `Square` should provide an implementation of `apply(int x)` that squares x.

The following are two important requirements for your implementation:

1. In order to avoid that the code runs forever, any part of the list should be evaluated only when necessary for any of the above methods of `List` to produce its answer.
2. In the interest of greater efficiency, each element of the list is to be computed at most once even if the user of the list requests the value more than once.
Question 3.3 continued:
Question 3.3 continued: