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) 9 marks

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

Regular languages

(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.

No.

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

A push-down automaton is a non-deterministic finite automaton augmented with a stack. The transition function maps state-character-character triples to state-string pairs. More precisely, \( \delta(q, x, y) = (q', \sigma) \) means: when reading character \( x \) in state \( q \) and with \( y \) on the top of the stack, transition to state \( q' \) and replace \( y \) with the string \( \sigma \).

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

Context-free languages

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

\( LL(k) \) languages, for any constant \( k \).
<table>
<thead>
<tr>
<th>(a) Static (lexical) binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>The binding between a name and the variable it refers to can be determined at compile time based on the nesting of language constructs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b) Dynamic binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>The binding between a name and the variable it refers to can be determined only at runtime because it is determined by the most recent scope that was entered but has not been left yet and which defines a variable with this name.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(c) Shallow binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>When passing a function ( f ) as an argument to another function, the free variables in ( f ) are bound only when ( f ) is invoked.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(d) Deep binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>When passing a function ( f ) as an argument to another function ( g ), the free variables in ( f ) are bound when ( f ) is passed to ( g ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(e) Applicative order evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>When passing an expression as an argument to a function, the expression is evaluated and the result is passed to the function.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(f) Normal order evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>When passing an expression as an argument to a function, the expression is passed to the function in unevaluated form and is evaluated only when its value is required.</td>
</tr>
</tbody>
</table>
Question 1.3 (Types) 9 marks

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

A type system is a mechanism for defining types and operations that be performed on objects of each type. It includes rules that specify type equivalence (are two types the same?), type compatibility (can a value of one type be used in a context where a value of a different type is expected?), and type inference (what’s the type of an expression, given the types of its components?). Types express intent and thus allow the compiler and/or runtime system to catch probably unintended uses of certain values whenever the rules of the type system are violated.

(b) What is name equivalence of types?

Two types are equivalent if and only if they have the same name, that is, if and only if they are the same type.

(c) What is structural equivalence of types?

Two types are equivalent if they have the same structure, that is, if they are composed of the same built-in types.

(d) What is a statically typed language?

All type checking is performed at compile time.

(e) What is a dynamically typed language?

All type checking is performed at runtime.
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.

The mark-and-sweep algorithm proceeds in three phases. The first phase marks every object on the heap as useless. The second phase uses the static variables and the variables on the stack as sources for a depth-first search of the reference graph of all objects. For every visited object, we explore all pointers it contains. For each such pointer, if the object it references is currently marked as useless, we mark this object as useful and visit the object recursively. The third phase releases all objects that remain useless after the second phase and compacts the useful objects to ensure there is no external fragmentation.
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>10</td>
</tr>
<tr>
<td>Static ✓</td>
<td>Dynamic ✓✓</td>
<td>0</td>
</tr>
<tr>
<td>Static ✓</td>
<td>Dynamic ✓✓</td>
<td>110</td>
</tr>
<tr>
<td>Static ✓</td>
<td>Dynamic ✓✓</td>
<td>100</td>
</tr>
</tbody>
</table>

Question 2.2 (Short-circuit evaluation) 6 marks

Consider the following Java code.

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.

This program throws a NullPointerException because the Boolean test succeeds and, thus, the program tries to print the result of b.f(), but b is null.

Assuming Java did not use short-circuit evaluation of Boolean expressions, what would the output be? Justify your answer.

The program throws a NullPointerException in this case as well. This time, however, the exception is raised when a.f() tries to invoke g() on a.z, which is null.
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**

```cpp
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();
}
```

11 seconds.
a is constructed using the default constructor; b using the copy constructor.

**Program 2**

```cpp
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();
}
```

22 seconds.
a and b are constructed using the default constructor. Then f() makes a copy of a using the copy constructor, which is then assigned into b using the assignment operator.
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)  

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.
Question 3.2 (Parsing) 5 marks

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>Prog $\rightarrow$ Stmt $$</td>
<td>{if,while,begin}</td>
</tr>
<tr>
<td>Stmt $\rightarrow$ if Expr then Stmt else Stmt</td>
<td>{if}</td>
</tr>
<tr>
<td>Stmt $\rightarrow$ while Expr do Stmt</td>
<td>{while}</td>
</tr>
<tr>
<td>Stmt $\rightarrow$ begin Stmts end</td>
<td>{begin}</td>
</tr>
<tr>
<td>Stmts $\rightarrow$ Stmt ; Stmts</td>
<td>{if,while,begin}</td>
</tr>
<tr>
<td>Stmts $\rightarrow$ $\epsilon$</td>
<td>{end}</td>
</tr>
<tr>
<td>Expr $\rightarrow$ id</td>
<td>{id}</td>
</tr>
</tbody>
</table>

parseProg:

```java
switch next symbol of
    case if | while | begin: parseStmt(); match($)
    default: raise Error
```

parseStmt:

```java
switch next symbol of
    case if: match(if); parseExpr(); match(then); parseStmt() match(else); parseStmt()
    case while: match(while); parseExpr(); match(do); parseStmt()
    case begin: match(begin); parseStmts(); match(end)
    default: raise Error
```

parseStmts:

```java
switch next symbol of
    case if | while | begin: parseStmt(); parseStmts()
    case end: // Do nothing
    default: raise Error
```

parseExpr:

```java
switch next symbol of
    case id: match(id)
    default: raise Error
```

match(sym):

```java
if next symbol equals sym then
    consume next symbol
else
    raise Error
```
Question 3.3 (Lazy evaluation) 20 marks

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 \([1, 2, 3, \ldots]\).
- `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 \(i\)th 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.

```java
public abstract class Function {
    public abstract int apply(int x);
}

public class Square extends Function {
    public int apply(int x) { return x*x; }
}
```
public abstract class List
{
    public static List intList() { return new HeadTailList(new Head(1),
        new Tail(2)); }

    public int nth(int i) { if (i==0) return head();
        else return tail().nth(i-1); }
    public List map(Function f) { return new HeadTailList(new Apply(f, head()),
        new Map(f, tail())); }

    public abstract int head();
    public abstract List tail();
    protected abstract List force();
}

private static class HeadTailList extends List
{
    public HeadTailList(Value h, List t) { _head = h; _tail = t; }

    public int head() { _head = _head.force(); return _head.value(); }
    public List tail() { _tail = _tail.force(); return _tail; }
    protected List force() { return this; }

    private Value _head;
    private List _tail;
}

private static class Tail extends List
{
    public Tail(int i) { _i = i; }

    public int head() { throw new Error("List should be forced first"); }
    public List tail() { throw new Error("List should be forced first"); }
    public List force() { return new HeadTailList(new Head(_i),
        new Tail(_i+1)); }

    private int _i;
}

private static class Map extends List
{
    Map(Function f, List l) { _f = f; _l = l; }

    public int head() { throw new Error("List should be forced first"); }
    public List tail() { throw new Error("List should be forced first"); }
    public List force() { return new HeadTailList(new Apply(_f, _l.head()),
        new Map(_f, _l.tail())); }

    private Function _f;
    private List _l;
}
private static abstract class Value
{
    public abstract Value force();
    public abstract int value();
}

private static class Head extends Value
{
    public Head(int x) { _x = x; }

    public Value force() { return this; }
    public int value() { return _x; }

    private int _x;
}

private static class Apply extends Value
{
    public Apply(Function f, int x) { _f = f; _x = x; }

    public Value force() { return new Head(_f.apply(_x)); }
    public int value() { throw new Error("Value should be forced first"); }

    private int _x;
    private Function _f;
}