Name:

Student number:

Please read the following instructions carefully:

- Fill in your name and student number above. Be prepared to identify yourself with your student card when you submit your exam.
- This is a closed-book exam. You are forbidden from accessing any external material, notes, electronic material, or online resources.
- Answer each open question in the space provided. Write clearly and legibly. Answer each multiple choice question by marking the space provided.
- A maximum of 100 points can be obtained by the questions of this exam, to be divided by 10 to yield the mark for the exam.

Please do not write in the space below.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple choice</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>DSLs and Metaprogramming</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Semantics</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Parallelism and Concurrency</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Multiple choice

In the multiple choice questions below, you can earn two points for each correct answer. If you fill in an incorrect answer one point will be deducted from your score. If you leave a question blank, no points will be deducted or awarded. These rules only apply to the multiple choice questions of this exam.

(a) (2 points) Which of the following statements is true?
   - To control access to mutable memory, Rust’s borrowing mechanism restricts how pointers and address information may be passed between methods.
   - To control access to mutable memory, Rust’s IO monad restricts side-effects to those functions that return a value in the IO monad.
   - To control access to mutable memory, Rust’s system of algebraic effects restrict when memory operations may be performed.
   - All of the above.

(b) (2 points) Both Elm and Purescript are both languages that compile to Javascript. Which of the following statements is true?
   - Like Javascript, both Elm and Purescript embrace dynamic typing.
   - Compiled Elm code is executed in a run-time system implemented in Javascript; Purescript aims to generate human-readable Javascript.
   - Both Elm and Purescript restrict effects using an IO monad, inspired by Haskell.
   - Elm and Purescript extend Javascript with a full blown object system, including inheritance and dynamic dispatch.

(c) (2 points) Which of the following statements is true?
   - Protocols in Swift define precisely how classes and objects defined in Objective-C may interface with Swift code.
   - Protocols are used to count the number of references to an object to facilitate garbage collection.
   - Protocols in Swift allow you to overload certain function or operator names.
   - Protocols in Swift restrict access to the network or shared data sources.

(d) (2 points) Which of the following statements is true?
   - OCaml allows you to organize code into modules; each module contains several methods, known as functors.
   - OCaml allows you to organize code into functors; each functor contains several methods and data types.
   - OCaml allows you to organize code into modules; in OCaml functors on the other hand, provide a uniform abstraction over data types that support a map operator, such as lists and trees.
   - OCaml allows you to organize code into modules; modules may be passed as arguments to functors to create new modules.

(e) (2 points) Which of the following statements is false?
   - Templates in C++ allow you to define generic or polymorphic functions that operate on many different types.
   - Templates enable a form of metaprogramming that C++ allows you to generate new code at compile time.
   - Templates in C++ are Turing complete, allowing programmers to run arbitrary computations at compile time.
   - Templates in C++ are statically checked to never produce ill-typed code.
(f) (2 points) Given the following Prolog definitions:

```prolog
foo(a).
foo(b).
foo(c).
bar(c).
bar(b).
baz(X) :- foo(X), bar(X).
```

What is the result of the query `baz(X)`.

○ X = a; X = b; X = c.
○ X = b; X = c.
○ X = c; X = b.
○ X = c; X = b; X = a.

(g) (2 points) Which of the following statements is *false*?

○ Like Purescript, Idris tracks the usage of different kinds of effects using an `Eff` monad;
○ Like Haskell and OCaml, Idris provides language support for interfacing with C code;
○ Idris’s dependent types allow values to depend on data types.
○ Idris’s powerful type system allows you to prove properties about your programs.

(h) (2 points) Which of the following statements is *false*?

○ Scala runs on the Java Virtual Machine (JVM), making it easy to interface with existing Java libraries.
○ Scala is a hybrid functional-object oriented language that supports many different styles of programming.
○ Scala’s traits allow you to define an interface; traits can be combined using mixins.
○ Like most object oriented languages, any class in Scala can only inherit from a single trait.

(i) (2 points) What is the result of evaluating the following expression in Racket:

```racket
(eval (first '((+ 1 2) (+ 3 4))))
```

○ `'(+(+ 1 2) (+ 3 4)))
○ 7
○ 3
○ a dynamic failure

(j) (2 points) Which of the following lambda terms could correspond to the expression `\(\lambda(\lambda0)1\)` written using De Bruijn indices:

○ \x \y \x \y \x \x
○ \y \x \x \x
○ \y \x \x \x
○ \x \y \x \x
(k) (4 points) Give two examples of how a programming language presented by one of the students during this course uses a specific *static semantics* to control or limit the *dynamic behaviour* of a program.

DSLs and Metaprogramming

(a) (2 points) Explain the difference between a *shallow* and a *deep* embedding of a domain specific language.

(b) (2 points) Why would you choose to *embed* a domain specific language, rather than implementing a new language from scratch? What are the drawbacks of doing so?
(c) (2 points) Briefly describe two different approaches for representing variable binding in an embedded domain specific language.

(d) (2 points) What is a *staged* programming language?

(e) (4 points) Template programming in C++ places severe restrictions on the programs that can be used to generate new code; Template Haskell is far more liberal, allowing arbitrary code to be executed. Explain the pros and cons of both these approaches using an example.
(f) (3 points) In this course, we have seen examples of metaprogramming in both Racket and Haskell. Why is Template Haskell so much more complicated than Racket’s `quote-eval` mechanism?

Semantics

For this question, we will consider a small expression language consisting of:

- boolean constants `true` and `false`;
- if-then-else expressions;
- the constant `0`;
- the successor operation `succ` that increments its argument;
- the `iszero` function that checks that returns `true` when its argument is `0` and `false` otherwise.

The values of this language consist of the boolean constants `true` and `false`, `0`, and any number of `succ` applications to `0`. 
We can give a dynamic semantics for the boolean fragment of the language as follows:

\[
\begin{align*}
\text{true} & \rightarrow t_2 & \text{IF-T} \\
\text{false} & \rightarrow t_3 & \text{IF-F} \\
\text{true} & \rightarrow t_1' & \text{IF} \\
\text{false} & \rightarrow t_1' & \text{IF} \\
\text{Stop} & \rightarrow^* \text{t} & \text{Step}
\end{align*}
\]

(a) (4 points) Explain the difference between static semantics and dynamic semantics.

(b) (5 points) Give a derivation showing that \(\text{if (if true then false else true) then true else 0} \rightarrow^* 0\)
(c) (6 points) Extend this dynamic semantics with rules for 0, succ, and iszero. Your definitions should align with the choices made by most programming languages.

(d) (5 points) In this expression language, there are two types of expressions Bool (booleans) and Nat (natural numbers). Suggest a static semantics for this expression language by defining a relation $e : t$, relating expressions $e$ and types $t$ (the language does not have any variables, so your relation will not need a typing environment $\Gamma$). Your definitions should align with the choices made by most statically typed programming languages.
(e) (4 points) Explain what the properties *progress* and *preservation* are and how they relate dynamic and static semantics.

(f) (5 points) Sketch how to prove the following statement:

For any expressions $t_1$, $t_2$, $t_3$, and for any value $v$,
when $\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow^* v$
either $t_2 \rightarrow^* v$ or $t_3 \rightarrow^* v$. 
Parallelism and Concurrency

(a) (4 points) Parallelism and concurrency are two different notions. Give a brief definition of each and explain how they are different.

(b) (4 points) Briefly sketch two different mechanisms for writing concurrent programs we have seen in this course and two different mechanisms for writing parallel programs that we have seen in this course.
(c) (2 points) What property do Erlang and Haskell share that makes both these languages suitable for parallelism?

(d) (2 points) Erlang claims to be designed for writing robust systems, yet the approach to error handling is 'Let it crash!' – explain this apparent contradiction.

(e) (2 points) What is the effect of linking two Erlang processes?

(f) (2 points) When parallelizing sequential code, you often need to control the size of the tasks that are to be run in parallel. Explain one approach for doing so in either Erlang or Haskell.

(g) (8 points) Consider the following definition of mergeSort in Erlang.

```
merge_sort([]) -> [];  
merge_sort([X]) -> [X];  
merge_sort(Xs) -> {Ys,Zs} = split(Xs),  
                  merge(merge_sort(Ys),merge_sort(Zs)).
```

Using spawn_link, self, and message passing, write a parallel Erlang version of merge_sort. You may reuse functions defined above in your answer without including their definitions. Ensure that the task granularity is not so fine that the overheads of parallelism dominate the run-time.
(h) (5 points) ‘Laziness is a blessing and a curse when it comes to parallelizing code.’ – what is meant by this statement? Illustrate your answer with examples and by referencing techniques for parallel programming that we have seen in the context of this course.