Cs358 Exam

Name: _____ ID: _____

This assignment has **6** questions, for a total of **75** marks.

Question 1: **Z combinator typing** ..... 5 marks

This is the Z combinator in ULC:

$$\lambda f.\,(\lambda x.\,f(\lambda y.\,((x\ x)\ y)))(\lambda x.\,f(\lambda y.\,((x\ x)\ y)))$$

Add type annotations as well as fold/unfolds and prove it can be typed in System F + isorecursive types. Its type is $((\tau_1 \to \tau_2) \to (\tau_1 \to \tau_2)) \to (\tau_1 \to \tau_2)$ for arbitrary τ_1 and τ_2.

Question 2: **Diverging with references** . . . . . 5 marks

Using references, there exists a (syntactically) well-typed closed term that (i) does not use roll nor unroll and (ii) diverges. Write such a term, prove that it is well-typed and show that it diverges.

Question 3: **Natural numbers-based heap** . . . . . 10 marks

In this exercise you will replace the abstract heap of System F with an assembly-like one. Drop all terms, types, typing rules, evaluation contexts and primitive reduction rules related to the heap with abstract locations. Add a heap that is a map from natural numbers to values ($H ::= \varnothing \mid H; n \mapsto v$). Add terms for allocating, reading and writing on the new heap (Hint: to start, identify the type of locations). Show typing rules for heap-related terms as well as COS rules for heap-related terms.

These additions must keep the language safe (normalisation is not achievable, as suggested by the first exercise), so argue why the additions are safe.

Question 4: **Formalising capability machines** ..... 20 marks

Capability machines extend assembly instructions with explicit capabilities such that reading and writing on memory is only allowed if a capability is provided.

Take ULC (with Nats, sums and products) and add a heap from natural numbers (i.e., as the one from assignment 6), make allocation deterministic starting from 0, each new location is at the next number. Extend the language with capabilities and formalise their semantics. You choose how to model them, choose wisely according to their behaviour as described below.

Capabilities are unforgeable and unobservable tokens which the program can create. Every time a memory location is created, it is unprotected. The language must provide primitives for protecting a location given a capability, this should only be possible if the location is unprotected. Reading and writing a memory location is always possible if the location is unprotected. However, if the location is protected with a capability, reading and writing that location is only possible if the same capability is provided at reading and writing time.

Question 5: **Thread-based concurrency** . . . . . 20 marks

Take STLC with sums and pairs. Add a new term for spawning a process, call that *fork t*. *fork t* must not reduce t, instead it must step to 0 and immediately spawn a thread whose body is t. Threads are scheduled nondeterministically for now, though you may want to look at the next exercise to build a more robust solution.

Formalise the statics and dynamic semantics of such a concurrent language and show all changes to the formalisation of the language.

Question 6: **Round-robin scheduling** . . . . . 15 marks

Take the language from the previous exercise. Change the scheduling to be round-robin. A thread executes for 10 steps and then control passes to the next thread (if there is one). If a thread terminates before 10 steps, control passes to the next thread. The next thread to be scheduled must be the next one in order of spawning.

Formalise the scheduling process and show changes to the formalisation of the language.

Make the formalisation elegant.