Section 1: Developing a type system

One of the challenges of writing multi threaded code in an imperative context is that memory can “change under your feet” if another thread concurrently writes to your memory location. In this exercise, we are going to develop a simple type system to help the programmer control which values can be updated by which threads, allowing programmers to better identify potential sources of non-determinism.

The basic syntax for our language is as follows:

\[
\begin{align*}
& \text{variable} \quad x \\
& \text{constructor} \quad \text{new } C( ) \\
& \text{field access} \quad e.f \\
& \text{variable declaration} \quad \text{let } x = e \text{ in } e' \\
& \text{field assignment} \quad e.f := e' \\
& \text{sequential composition} \quad e_1; e_2 \\
& \text{thread creation} \quad \text{fork } e
\end{align*}
\]

In order to control which variables can be updated by which threads, we will separate objects into two categories: Local and Global objects, identified by the letters \( \mathcal{L} \) and \( \mathcal{G} \) respectively. When an object is created, we can create local (resp. global) objects by invoking \( \text{new}<\mathcal{L}> C() \) (resp. \( \text{new}<\mathcal{G}> C() \)).

Global objects can be read and written by any thread; local objects can be read by anyone, but can only be written to by the thread that owns them. This partitioning of objects means that we need to have three different types of references: a) Global references can point to global objects, b) Local references can point to local objects owned by the current thread, and c) Remote references which point to local objects owned by another thread. We will identify the three types of references with the letters G, L and R respectively. If we have a local reference, we can read and write to the object pointed to by the reference, and we know the object will never change under our feet. Global references also allow us to read and write to the object pointed at by the reference, but we need to be careful, because other threads can do the same. Remote references allow us to observe the local state of another thread, but we cannot write to an object through a remote reference.

Part 1: Typing Variables

For this part, we will ignore fields so we can concentrate on how the type system treats local variables, object creation and fork. Our typing rules will be defined using an environment \( E \) that maps each variable to one of three types: G, L and R (for global, local and remote).
Homework 3: Types and Axiomatic Semantics  Sections 1 and 2a due
November 3, 1:00 PM; 2b due Nov 8, 1:00 PM.

a) Complete the rules shown below (10 pts total):

\[
\begin{align*}
(x:\tau) \in E & \quad \frac{x}{E \vdash x : \tau} \\
new < L > C(\_): \underline{\_} & \quad \text{new } < G > C(\_): \underline{\_}
\end{align*}
\]

b) Define E’ for each of the rules below and complete any rules that need completing (5pts each):

\[
\begin{align*}
E \vdash e_1: \tau_1 & \quad E' \vdash e_2: \tau_2 \\
E \vdash e_1; e_2: \underline{\_}
\end{align*}
\]

\[
\begin{align*}
E \vdash e_1: \underline{\_} & \quad E' \vdash e_2 \underline{\_} \\
E \vdash \text{let } x: \tau = e_1 \text{ in } e_2: \underline{\_}
\end{align*}
\]

\[
\begin{align*}
E' \vdash e : \tau & \\
E \vdash \text{fork } e : \text{unit}
\end{align*}
\]

Part 2: Fields.

For fields, we have the following two type annotations:

G: This field will always point to a global object.
R: This field will always point to an object local to some thread (not necessarily the current one).
S: This field will always point to an object in the same space as this object. E.g. if this object is global, then an S field points to another global object, and if this object is local to a thread t, a field S will point to another object local to thread t.

a) (10pts) Write the typing rules for the expression e.f (make sure you consider all the cases for the type of e and of field f).

b) (15pts) When assigning e.f := e’, consider all combinations of types for e, e’ and f, and describe for which combinations the assignment should be rejected by the type system.
Section 2: Axiomatic Semantics

a) For each of the loops below, identify the loop invariants that are necessary to prove the sentence. Be sure to justify your answer (10 pts each).

\[ \{ y = y_0, k = k_0, t = y_0 - k_0 \} \text{while}(t > 0) \{ y = y - 1; t = t - 1; \} \{ y \leq k_0 \} \]

\[ \{ y < \text{len}, \text{Err} = 0 \} \text{while}(q < y) \{ \text{if}(q > \text{len}) \{ \text{Err} := 1 \}; q := q + 1; \} \{ \text{Err} = 0 \} \]

\[ \{ a = 0 < b = N \} \{ A[0] < q < A[N] \} \text{while}(b - a > 1) \left\{ \begin{array}{l} t := \frac{a + b}{2}; \\
\text{if}(A[t] > q) \{ b := t; \} \text{else} \{ a := t; \} \end{array} \right\} \{ A[a] \leq q \leq A[a + 1] \} \]

b) Prove that if the input to this program is a non-empty list, its output will be another list with two copies of every element from the input list (20 pts):

```java
void duplicate(List in){
    n = in.head;
    while(n!= null){
        t = new Node();
        t.next = n.next;
        n.next = t;
        n = t.next;
    }
}
```