Programming Language Concepts

Mid-Semester Examination, II Semester, 2023–2024
Practice Questions

- *Answer all questions.*
- *In any question that asks for code, provide Java-like or Rust-like pseudocode, as appropriate. Syntax errors will be ignored, provided the code is conceptually correct.*
- *Supply explanations for any code you write, ideally as annotations alongside the code.*

---

1. Consider the program skeleton on the right.

   Suppose we initially call the function `M`, so that the stack has only the activation record (AR) corresponding to `M()`.

   (a) Is there a theoretical limit on how high the stack can get (i.e. how many ARs the stack will hold at one time) in the course of executing `M()`?

   (b) Suppose the stack has grown to height 7. Can you draw the contents of the stack? You do not need to display the contents of each AR. Just display the name of the function whose AR it is, and draw the control and access links.

   (c) We again consider an activation stack of height 7, where the first call is to `M()`. Consider the latest function invocation (whose AR is on top of the stack). Assume the ARs are numbered 1 to 7, with 7 being the bottom (containing the AR of `M()`). Which of the variables (among `a`, `b`, `m`, `n`, `x`, `y`, `z`) are in the scope of that function invocation? Which of the variables are alive? For each variable in scope, mention the AR that gives its value.

```
int M() {
  int a;
  int P(int n) {
    int x, y, z;
    int Q() {
      int b;
      int R(int m) {
          ...
          z = P(m);
          ...
      }
      ...
      y = R(b);
      ...
    }
    ...
    x = Q();
    ...
  }
  ...
  return P(a);
}
```

2. The Siruseri public library has an online access system for member services. All information about books and users is stored in a single object of class `LibraryData` using private instance variables. This class supports functions for online enquiries and reservations. The corresponding functions have the following (incomplete) signatures:

```
... boolean check_availability(String title);
// is a book available for issue?
... boolean reserve_book(String title);
// make a reservation: return value confirms status of reservation
```

   To access an account, a user must login with a valid username and password. After logging in, the user can perform a single *transaction*. A transaction consists of upto three availability enquiries and one book reservation. To invoke another transaction, the user must log in again. There is no restriction on a user logging in multiple times in parallel. Each successful login in parallel can separately perform one transaction.

   Logging into the account is implemented by a function of the form

```
... login(String u, String p);
// log in as user u with password p
```

Explain how to design the class `LibraryData` to provide the required single-transaction-per-login access to users. Describe your design in terms of Java-like pseudocode—give details of interfaces, data definitions and and function signatures. For function bodies, you can write comments instead of detailed pseudocode to explain the features of your design.

3. Here is a skeleton definition in Java of a class `List` that implements a generic linked list of `Object`.

   Rewrite this skeleton to using type variables so that `List` is a parameterized class and each instance of `List` can store any value that is a subtype of the concrete type supplied when instantiating `List`.

   Note: You only need to provide the function signatures in the corresponding skeleton, not the detailed code for each function.

```
public class List {
  private Node head;
  ...

  public boolean isEmpty(){...}
    // Is the list empty?
  public Object remove_head(){...}
    // Remove and return first item in list
  public void append(Object o){...}
    // Add an item at the end of the list

  private class Node{
    private Object o;
    private Node next;
    ...
  }
}
```

4. Explain the difference between these two functions in Rust.

```
fn main1(){
    let mut s = String::from("PLC");

    let sref1 = &s;
    println!("sref1 = {}",sref1);

    let sref2 = &mut s;
    println!("sref2 = {}",sref2);
}
```

```
fn main2(){
    let mut s = String::from("PLC");

    let sref1 = &s;
    let sref2 = &mut s;

    println!("sref1 = {}",sref1);
    println!("sref2 = {}",sref2);
}
```