Ken Jin's answers version

National University of Singapore

CS2104 — Programming Language Concepts

AY2023/2024 Semester 1

Final Assessment

Time allowed: 2 hours

QUESTION PAPER

INSTRUCTIONS

1. This **QUESTION PAPER** contains **15** Questions in **8** Sections, and comprises **9** printed pages, including this page.
2. The **ANSWER SHEET** comprises **8** printed pages.
3. Use a pen or pencil to **write** your **Student Number** in the designated space on the front page of the **ANSWER SHEET**, and **shade** the corresponding circle **completely** in the grid for each digit or letter. DO NOT WRITE YOUR NAME!
4. You must **submit only** the **ANSWER SHEET** and no other documents. Do not tear off any pages from the ANSWER SHEET.
5. All questions must be answered in the space provided in the **ANSWER SHEET**; no extra sheets will be accepted as answers.
6. Write legibly with a **pen** or **pencil** (do not use red color). Untidiness will be penalized.
7. For **multiple choice questions (MCQ)**, **shade** in the **circle** of the correct answer **completely**.
8. The full score of this assessment is **60** marks.
9. This is an **Open-Book** assessment. You are allowed to use any printed or written material.

Section A: Syntax [8 marks]

The syntax of the language Scheme consists of nested brackets called S-expressions. An example Scheme program looks like this:

```
(defun factorial (x)
   (if (zerop x)
       1
       (* x (factorial (- x 1)))))
```

(1) [2 marks] (MCQ)

How many tokens does the program above consist of?

A. 13
B. 14
C. 27
D. 52
E. 87

C. Count the number of tokens - E.g. "(", "defun" "factorial" "(" "x" ")" is 6
Generally, whitespace don't count as tokens in Scheme.

(2) [2 marks] (MCQ)

On a scale from 1 (very easy) to 5 (very difficult), how do you judge the difficulty of parsing Scheme, compared to languages such as Python and JavaScript?

A. 1
B. 2
C. 3
D. 4
E. 5

Any number works if you can justify it. However, I will pick 1 or 2 as Scheme is easier to parse (for me) than Python.

(3) [4 marks]

Give a short argument (two or three sentences) for your judgment in Question (2).

[Answer based on what I chose above, your answer can differ]. However, note that in Python, there is whitespace significance and as of version 3.10 onwards cannot be parsed with a LL(n) parser (it uses a PEG parser). While Scheme can be parsed with a simple LL(n) parser.

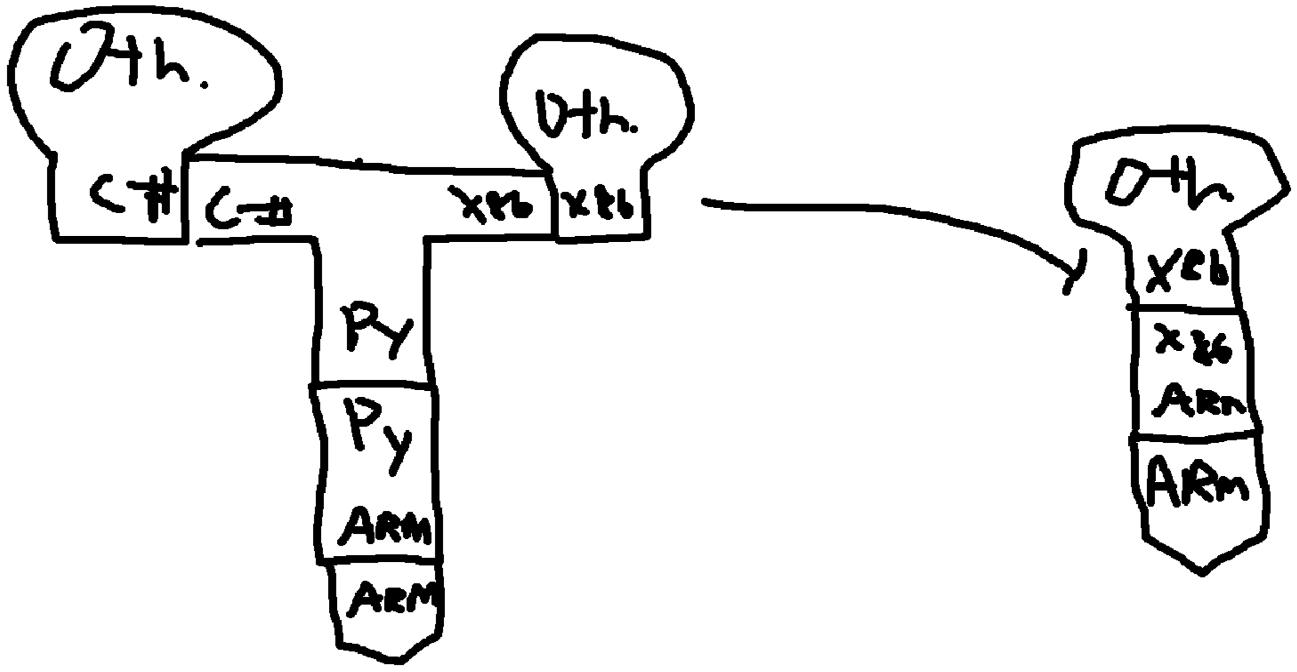
Section B: Interpreters and Compilers [8 marks]

(4) [8 marks]

You want to run program "Othello" written in C# and the only computer at your disposal has an ARM processor. You are given a compiler from C# to X86 machine code written in Python. You are also given a Python interpreter written in ARM machine code, and an emulator for X86 machine code written in ARM machine code.

Draw all the T-diagrams of the programming language implementation steps that are needed to run "Othello" on the given computer.

Sorry for the bad handwriting :(.



Section C: Types [7 marks]

Option types are used to handle the absence of a value in a safe way. They are a better alternative to using null references, which can lead to runtime errors if not handled carefully.

An option type has two variants:

- Some: Represents the presence of a value. For instance, `Some(x)` where `x` is the actual value.
- None: Represents the absence of a value.

In functional languages like Haskell, SML or OCaml, option types are used extensively for functions that may or may not return a value. They encourage the programmer to explicitly handle the case of missing values.

(5) [3 marks]

Declare a float-option data type in your favourite functional language. A float option should represent an "optional" floating point number: the presence of a specific floating point number or the absence of any floating point number. In your declaration, you may not use any existing data types other than `float`.

```
Ocaml:type float_option =  | Float of float  | Nil
```

(6) [4 marks]

Using your float-option data type, write a function `accessFloatArray` in your favourite functional language that takes an array of floating point numbers and an integer as arguments, and returns a float option as the result of accessing the given array at the given index. You may assume that array indices start at 0 and that the unary function `arrayLength` returns the length of a given array as an integer.

Use **explicit** type declarations for the argument types and the return type for `accessFloatArray`.

```
open Array

let accessFloatArray (arr: float array) (idx: int): float_option  =
  if idx >= arrayLength arr then Nil else Float arr.(idx)
```

Section D: Memory Management [11 marks]

The lectures have presented and compared the heap management techniques of reference counting, copying garbage collection and mark-sweep garbage collection.

(7) [5 marks]

Write a function `main(n)` in pseudo-code that dynamically generates data structures during its execution. The function should be written such that copying garbage collection will be significantly more efficient than both mark-and-sweep and reference counting as `n` increases. Calling the function will continue to work efficiently even for large `n` when copying garbage collection is used, whereas systems that use reference counting or mark-sweep garbage collection will be significantly slower than copying garbage collection, or run out of memory to compute `main(n)` when `n` exceeds a certain value.

(8) [3 marks]

Explain in up to three sentences why reference counting behaves worse than copying garbage collection on your program.

(9) [3 marks]

Explain in up to three sentences why reference counting behaves worse than reference counting on your program.

Note: this tests your knowledge of characteristics of copy gc/mark-sweep/refcounting.
For a recap, look at my slides. A brief recap:
1. Copy GC over mark-sweep: has compaction, and performs better at low residency
2. Copy GC vs refcounting: Copy GC can clear cyclic data structure, refcounting cant.

An example program, in Python (assuming Python uses a linear bump allocator):

```
class Junk:
   def __init__(self):
      # Form a cycle that can't be cleared by refcounting!
      self.other = self

def main(n):
   root = []
   to_delete = []
   for i in range(n):
      if i % 2:
         root.append(Junk())
      elseS:
         to_delete.append(Junk())
      # Randomly delete to_delete once in a while, to induce fragmentation, and
      # keep residency low
      if not i % 100:
         to_delete = []
      # Randomly delete root once in a while, to induce fragmentation, and keep
      # residency low.
      if not i % 300:
         root = []
```

Section E: Concurrency [6 marks]

In the course, you have seen how higher-level synchronization techniques such as semaphores can be built from lower-level techniques such as test-and-set. In this question, we explore an alternative low-level synchronization technique called "compare-and-swap" (CAS).

You can assume that the CAS operation is atomic and works as follows:

`CAS(pointer, oldValue, newValue)` - This operation checks if the value at `pointer` is equal to `oldValue`. If yes, it atomically sets the value at `pointer` to `newValue` and returns true. If the value at pointer is not equal to `oldValue`, it does nothing and returns false.

Semaphores which allow an arbitrary resource count are called *counting* semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called *binary* semaphores and are used to implement locks.

(10) [6 marks]

Implement a binary semaphore based on `CAS`. Requirements:

- Initialization: Describe how to initialize your semaphore.
- `Wait():` Implement the wait operation, ensuring that it properly decrements the semaphore and waits if the semaphore value is 0.
- `Signal():` Implement the signal operation, which increments the semaphore and wakes a waiting process if necessary.

```
int binary_sema = 1;

void Wait(int *ptr) {
  while (!CAS(ptr, 1, 0)) {}
}

void Signal(int *ptr) {
  // Note: this is safe over *ptr += 1
  // because signalling a binary semaphore with 1
  // is undefined behavior.
  CAS(ptr, 0, 1);
}
```

Section F: Parallelism [6 marks]

Consider the following pseudo-code for adding two matrices into a third matrix, assuming that they all have the same number of rows and columns and contain numbers:

```
procedure add(Matrix A, Matrix B, Matrix Result)
    for i from 1 to A.rows do
        for j from 1 to A.columns do
            Result[i, j] += A[i, j] + B[i, j])
        end for
    end for
end procedure
```

(11) [2 marks] (MCQ)

Which of the loops in this function are embarrassingly parallel?

A. none
B. both
C. Only the outer loop
D. Only the inner loop

(12) [4 marks]

Justify your answer in Question (11) with at most three sentences.

Note: again another question that could go either way, just make sure your answer is justified.
I choose B., both loops are embarassingly parallel.
My reasoning is that the operations in the loop have no data dependency (Result[i, j] does not depend on a previous Result[i, j], neither for A or B). An optimizing compiler will be able to parallelize the loop easily (assuming alignment and the other usual stuff).

Section G: Logic Programming [8 marks]

Consider the following Prolog program:

```
nth(0, [H|_], H).
nth(N, [_|T], X) :-
    nth(N1, T, X), N is N1 + 1.
```

What is the result of the following Source §3 program?

(13) [4 marks]

Predict the behaviour of the following query:

```
?- nth(2, [1, 2, 2, 3, 4, 5], X).
```

The prolog program above searches for the nth element in an array.

2th element of the array given produces 2. Querying repeatedly produces no new results, as that is the only solution.

State if this query succeeds, and if yes, what binding of X it produces. State what happens upon repeatedly requesting new solutions using the semicolon, a feature of interactive Prolog systems to enforce backtracking. Explain your prediction with two or three sentences.

(14) [4 marks]

Predict the behaviour of the following query:

```
?- nth(I, [1, 2, 2, 3, 4, 5], 2).
```

The query produces 1 then 2 upon repeated querying then terminates.
This is searching for the indexes which contains the element "2", which is 1 and 2.

State if this query succeeds, and if yes, what binding of I it produces. State what happens upon repeatedly requesting new solutions using the semicolon, a feature of interactive Prolog systems to enforce backtracking. Explain your prediction with two or three sentences.

Section H: Scope [6 marks]

Python and JavaScript are both lexically (statically) scoped languages, but they handle the scope of declarations differently. While Python uses function scope for its local declarations, JavaScript typically uses block scope.

To explore this difference, consider two versions of JavaScript:

1. **JavaScript (JS):** The standard version with block scope, where **let** and **const** declarations are scoped to the closest enclosing block.

2. **JavaScript with Function Scope (JSF):** A hypothetical variant where the scope of **let** and **const** declarations within a function extends to the entire function body, regardless of blocks.

(15) [6 marks]

Write a short program that demonstrates the difference in scoping rules between JS and JSF. Your program should:

- Include at least one function.
- Use `let` or `const` declarations.
- Behave differently when run in JS and JSF due to the difference in scoping rules.

Requirements:

- Clearly comment your program to explain how the scoping rules of JS and JSF lead to different behaviours.
- Ensure the program is valid and would execute without errors in both JS and JSF environments.
- The difference in behaviour should be clearly observable in the output or side effects of the program.

I can't think of an answer that fulfils all the reuirements,
and I consulted Prof Martin and he wasn't able to come up with one at the time too.
So if you can come up with one please let me know. Thank you!

```
function f() {
   { let x=0; }
   try { return x; }
   catch { return 1;}
}
```

——— END OF QUESTIONS ———