CS131
PROGRAMMING LANGUAGES
(WEEK 10)

UCLA WINTER 2019

TA : SHRUTI SHARAN

DISCUSSION SECTION : 1D

ANNOUNCEMENTS

- **Today** is the last day to submit homeworks! No late submissions after **11:55PM**
- Course evaluations due **Tomorrow 8 AM**
  - Please fill it - it really helps improve the course for next quarter.
- Final exam **Thursday 8:00 AM – 11:00 AM**
  - Open book, open notes
    - Might be beneficial to print out your homeworks / solutions for reference.
  - No computers/gadgets/digital devices allowed.

SAMPLE QUESTIONS FOR FINALS

QUESTION 1

(5 minutes). Why is heap fragmentation not a problem in a traditional Lisp implementation where "cons" is the only heap storage allocation primitive?

QUESTION 2

What are the types of the following functions and what do they do:

a) let s x = x + 1

b) let rec i min max =
```
    if max < min then []
    else min :: i (s min) max
```

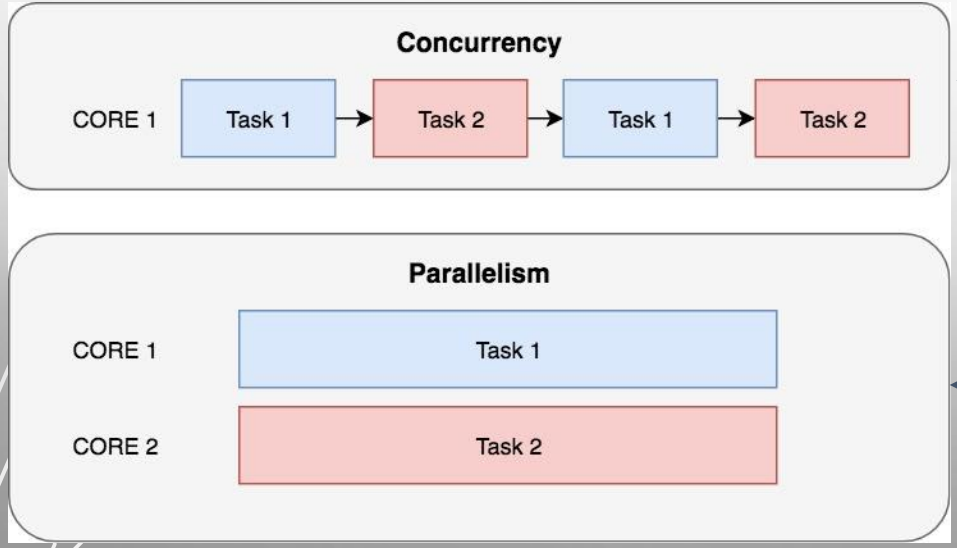
c) let rec f p = function
```
  | []  -> []
  | a::r ->
     let fpr = f p r in if p a then a::fpr else fpr
```

d) let x n =
```
    f (fun m -> not (m mod n =0))
```

e) let t max =
```
    let rec f1 = function
    | [] -> []
    | n::r ->
       n :: if max < n*n then r else f1 (x n r)
    in
    f1 (i 2 max)
```

QUESTION 3

This question asked about an old framework used in previous Python homeworks - Make sure you know how asyncio works, how it can handle multiple simultaneous connections, etc



Python way

Not supported by default Python implementation (without running multiple processes)

QUESTION 4

The programmers of the world are still using the first major imperative language (Fortran), the first major functional language (Lisp), and the first major logic-programming language (Prolog), but they no longer use the first major object-oriented language (Simula 67).

That is because object-oriented languages are fundamentally different. Here's why:

If you agree with this argument, finish it by adding some justifications and examples. If not, argue the contrary.

QUESTION 5A

Consider the static-scoping semantics given in pages 512-513 of Webber. (508-509 in the 2nd edition)

Can these semantics be implemented easily on a machine in which all storage is allocated on the stack, or do they assume the existence of a heap? If the former, give a nontrivial example and show why the heap is not needed for it; if the latter, give an example of why a heap is necessary.

QUESTION 5B

This question is the same as (a), except use the dynamic-scoping semantics

(Question 5a: Can these semantics be implemented easily on a machine in which all storage is allocated on the stack, or do they assume the existence of a heap? If the former, give a nontrivial example and show why the heap is not needed for it; if the latter, give an example of why a heap is necessary.)

QUESTION 6

Generally speaking, languages that use call-by-name or call-by-need parameter-passing conventions avoid hangs and/or crashes that would occur if the same program were evaluated using call-by-value. Can you think of any counterexamples ? That is, can you think of a program that has well-defined behavior with call-by-value, but which hangs with call-by-name or call-by-need ?

If so, give a counterexample; if not, explain why it's not possible to give a counterexample.

QUESTION 6 - CALL-BY-NAME HANGS

```
int count = 0; //global state
int bar() {
  count++;
  if (count > 1) {
    while (true) { ... }
  }
  return 1;
}
void foo(int a) {
  printf("%d", a);
  printf("%d", a);
}
foo(bar());
```

QUESTION 6 - CALL-BY-NAME/NEED HANGS

```
bool __is_foo_called = false; //global state
int bar() {
  if (__is_foo_called) {
    while (true) { ... }
  }
  return 1;
}

void foo(int a) {
  __is_foo_called = true;
  print("%d", a);
}

foo(bar());
```

QUESTION 7A - WRITE IN PROLOG

shift_left(L, R) succeeds if R is the result of "shifting left" the list L by 1. The leading element of L is lost.

For example, shift_left ([a,b,c], [b,c]).

QUESTION 7B - WRITE IN PROLOG

shift_right(L, R) is similar, except it shifts right.

For example, shift_right([a,b,c] , [a,b]).

QUESTION 7C - WRITE IN PROLOG

shift_left_circular(L, R) is like shift_left, except the leading element of L is reintroduced at the right.

For example, shift_left_circular([a,b,c] , [b,c,a]).

QUESTION 7D - WRITE IN PROLOG

shift_right_circular(L, R) is similar, except it shifts right.

For example, shift_right_circular([a,b,c] , [c,a,b]).

QUESTION 8

What is the set of possible behaviors for the following Scheme expression:

(car (begin (set! car cdr) (list car cdr)))

QUESTION 9

The abstract syntax for Scheme in R5RS section 7.2.1 lists only constants, identifiers, procedure calls, lambda, if, and set!. The following expressions aren't any of these things, so why are they valid expressions in Scheme? Or if they're not valid expressions, say why not.

```
(let ((a '())) a)

(cond ((eq? '() '()) 1) (#t 0))

(1)
```

QUESTION 10

This problem asked to convert homework-related Java code to Python - the exact problem is not relevant, but you should be able to convert code from one language to another and comment on the limitations

Example limitations when converting Java to Python:

- Static vs dynamic typing
- Field/method visibility - In Python everything is visible
- Java libraries might not have Python equivalents
- Multithreading
- No interfaces in Python

QUESTION 11

Explain the performance gains of the following language rule (if there are any):

- Arrays are origin-zero, not origin 1
- Arrays must be allocated statically or on the stack; they cannot be allocated on the heap
- Each object in an array must have the same size.
- Subscript violations result in undefined behavior; an exception may or may not be raised.

QUESTION 11 (CONT)

Explain the performance gains of the following language rule (if there are any):

- The array must have just one dimension
- The number of elements in an array is fixed at compile-time.
- Arrays must always be accessed directly; there are no pointers to arrays, and arrays cannot be passed by reference to procedures.

QUESTION 12

Suppose we add a new statement "return[N] EXPR;" to our C implementation, as an extension to C. N must be an integer constant expression with a nonnegative value, and EXPR can be any valid C expression. This new statement acts like "return EXPR;", except that it causes the Nth caller of the current function to return with the value of the expression EXPR. For example, suppose F calls G which calls H, and suppose H then executes "return[2]5;". This causes F to return immediately, with the value 5; G and H are silently exited in the process. By definition, "return[0]EXPR;" is equivalent to "return EXPR;".

What problems do you see in implementing "return[N]EXPR;"? Are these problems inherent to C, or can they be worked around by changing the semantics of the new statement slightly? Would it be easier to implement this new statement in some of the other languages that we have studied? If so, which ones and why? If not, why not?

GOOD LUCK WITH THE EXAM!

(P.S. REMEMBER TO SUBMIT THE COURSE EVALUATIONS TODAY)