Midterm Exam Review

Programming Language Theory

After Midterm

- We will study various programming language paradigms.
- For the first two weeks, we will go through ***Scripting Languages*** and ***OOP Languages*** quickly.
- The main dishes are ***Functional*** and ***Logic*** Programming Languages.
    - Functional Programming: Scala
    - Logic Programming: Prolog
- Practices + Assignments

This Week's Topics

- **Midterm Exam Review**
- PL Paradigm Overview
- Scripting Language

New Tablet's Arrived!

Question 1 (10pt). Please answer the following questions by either fill in the blanks, or write a simple sentence.

(a) (2pt) An interpreter repeats the three tasks, which are known as $\boxed{\textit{Read}}$ - $\boxed{\textit{Eval}}$ - $\boxed{\textit{Print}}$ -*Loop*, in short, REPL.

(b) (2pt) A system is $\boxed{\textit{Turing Complete}}$, if it can be used to simulate a Turing Machine.

(c) (2pt) What is a set of bindings at a specific point in the program at run-time?

$\boxed{\textit{Referencing Environment or Environment.}}$

(d) (4pt) Write a prefix expression equivalent to a mathematical equation $a - 5 \times (b - c)$. Your answer must be evaluated in the same order as the given equation (i.e., $b - c$ is computed first).

$\boxed{- \ a \times 5 - b \ c \ \textbf{or} \ - \ a \times - \ b \ c \ 5}$

How to Convert it?

$a - 5 \times (b - c)$

- Note that this infix expression is not evaluated from left to right, due to mathematical convention.

- v1 = b - c

  v2 = 5 * v1

  v3 = a - v2

- Generating a parse tree based on this, and visit the nodes in pre-order.

$- \; a \times - \; b \; c \; 5$

or

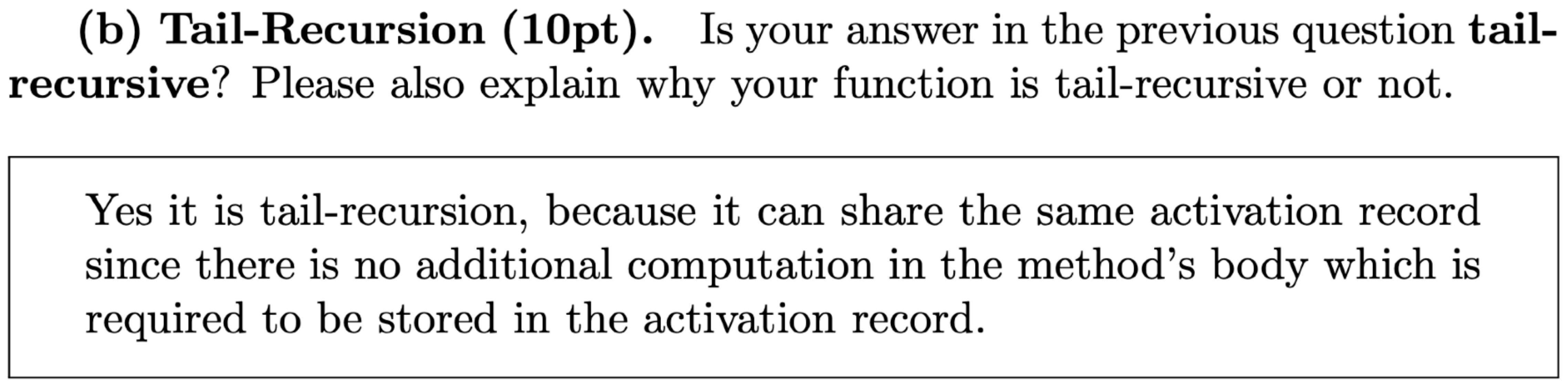
$- \; a \times 5 - b \; c$

Question 2 (20pt). Euclidean algorithm is an algorithm to compute the Greatest Common Divisor (GCD) of two integers a and b, $a > b$. We can denote GCD of `a` and `b` as `gcd(a, b)`. Then, `gcd(a, b)` satisfies the condition that, `gcd(a, b)` = `gcd(b, r)` if `a % b = r` and `r != 0`.

(a) Recursion (10pt). Complete the following recursive function which computes `gcd(a, b)` for two integers `a, b`. You can assume that `a > b` is guaranteed.

```
int gcd(int a, int b) {
        if( b == 0 ) {
                return  a ;
        }else{
                return  gcd(b, a%b) ;
        }
}
```

Tail-Recursion



(b) Tail-Recursion (10pt). Is your answer in the previous question **tail-recursive**? Please also explain why your function is tail-recursive or not.

Yes it is tail-recursion, because it can share the same activation record since there is no additional computation in the method's body which is required to be stored in the activation record.

- No additional computation.
- Share single activation record.
- Recursive call is the last thing in the function.
- Any one of the above comments.

Question 3 (20pt). Define a language L generated by the given context-free grammar G, by completing the conditions.

$$\begin{aligned} G =& (\{S, A, B\}, \{a, b\}, S, P), \\ & S \to Ab|aB \\ & A \to aaA|\varepsilon, \\ & B \to Bbb|\varepsilon. \end{aligned}$$

$$L(G) = \{a^n b^m : \boxed{\quad (a) \quad} \text{ or } \boxed{\quad (b) \quad} \}$$

(a): $\boxed{n = 2k, k \geq 0, m = 1}$

(b): $\boxed{m = 2k, k \geq 0, n = 1}$

$$\begin{aligned} G = &(\{S, A, B\}, \{a, b\}, S, P), \\ &S \to Ab|aB \\ &A \to aaA|\varepsilon, \\ &B \to Bbb|\varepsilon. \end{aligned}$$

$S \to Ab \to aaAb \to aaaaAb$

$S \to aB \to aBbb \to aBbbbb$

$S \to Ab \to b$

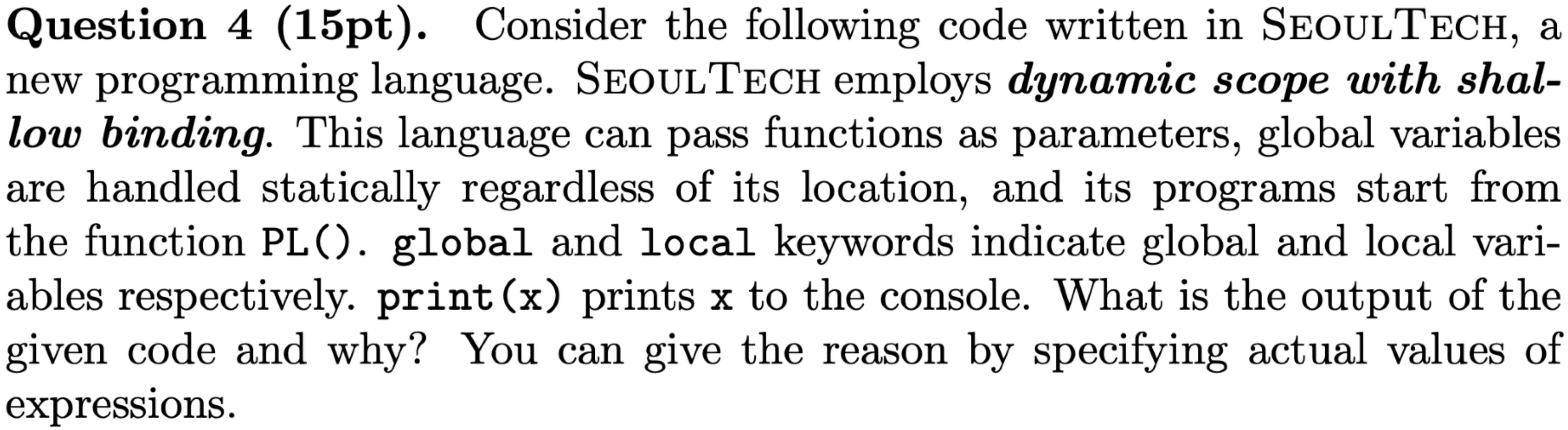
$S \to aB \to a$

$a^n b^m$: n is *even* number, m = 1

$a^n b^m$: m is *even* number, n = 1

$a^n b^m$: n=2k, k>=0, m = 1

$a^n b^m$: m=2k, k>=0, n = 1



Question 4 (15pt). Consider the following code written in SeoulTech, a new programming language. SeoulTech employs ***dynamic scope with shallow binding***. This language can pass functions as parameters, global variables are handled statically regardless of its location, and its programs start from the function `PL()`. `global` and `local` keywords indicate global and local variables respectively. `print(x)` prints `x` to the console. What is the output of the given code and why? You can give the reason by specifying actual values of expressions.

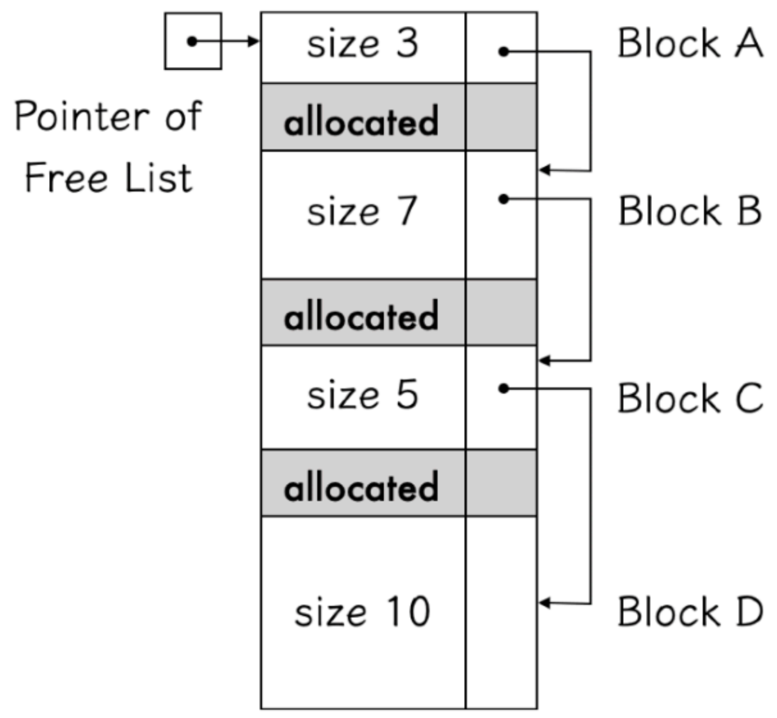
- Dynamic Scope with Shallow Binding.
- Can pass functions as parameters.
- Static Management of Global variables.
- Starts from PL().

Dynamic Scope and Shallow Binding

```
global x = 2;
func f(k) {
        return x+k;
}
func foo(g) {
        local x = 5;
        return x + g(1);
}
func bar(h, m) {
        return h(f) + m;
}
func PL() {
        print(bar(foo, 1));
}
```

$$\text{bar(foo, 1)} = \text{foo(f)} + 1 = (5 + \text{f}(1)) + 1 = (5 + (5 + 1)) + 1 = \mathbf{12}$$

Question 5 (15pt). This image shows the heap dynamically managed with variable length block method. Currently there are four free blocks Blocks A (size 3), B (size 7), C (size 5) and D (size 10), in the order attached in the free list. Now, answer the following questions. Note that (a)'s allocation doesn't affect to (b).

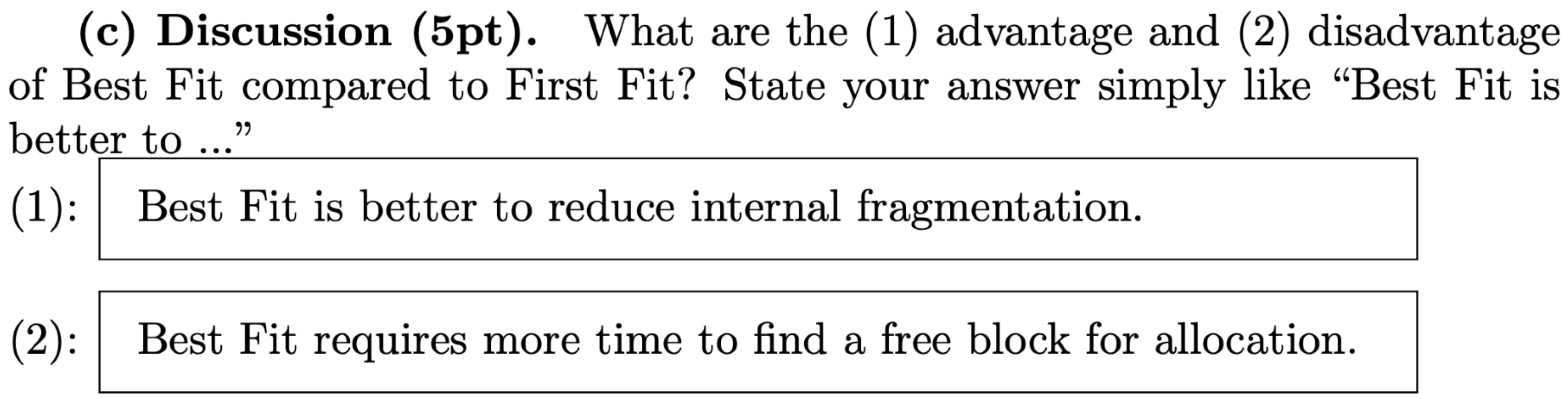


(a) First Fit (5pt). Suppose we are using *First Fit* strategy. When there is a memory request of size 4, which free block will be allocated?

$\boxed{\text{Block B}}$

(b) Best Fit (5pt). Suppose we are using *Best Fit* strategy. When there is a memory request of size 4, which free block will be allocated?

$\boxed{\text{Block C}}$



(c) Discussion (5pt). What are the (1) advantage and (2) disadvantage of Best Fit compared to First Fit? State your answer simply like "Best Fit is better to ..."

(1): Best Fit is better to reduce internal fragmentation.

(2): Best Fit requires more time to find a free block for allocation.

- (1): Internal fragmentation, Waste Memory, etc.
  - Something related to **Space**.
- (2): Need more time or slow, etc. - "allocation" or "find free blocks" or "processing allocation", etc.
  - Something related to **Time** or **Speed**.

Question 6 (20pt). Here is an image of Central Referencing Table (CRT) with a hidden stack to implement dynamic scope. On the left side, Blocks A, B, C and D are presented, and they are executed in that sequence. Fill in the blanks if necessary, to simulate dynamic scope implementation with CRT in this execution. When you fill the hidden stack, fill it from the bottom, since it is a "stack". Note that some of the blanks should be empty.

```
A:{
    x, y = ...
    B: {
        v, w = ...
    }
    C: {
        w, x = ...
        D: {
            y = ...
        }
    } //end of C
} //end of A
```

Block Execution

A

x	1	a1
y	1	a2
v	0	
w	0	

AB

x	1	a1
y	1	a2
v	1	b1
w	1	b2

ABC

x		
y		
v		
w		

ABCD

x		
y		
v		
w		

hidden stack

AB

ABC

ABCD

Block Execution

```
A:{
    x, y = ...
    B: {
        v, w = ...
    }
    C: {
        w, x = ...
        D: {
            y = ...
        }
    } //end of C
} //end of A
```

A

x	1	a1
y	1	a2
v	0	
w	0	

AB

x	1	a1
y	1	a2
v	1	b1
w	1	b2

ABC

x	1	c1
y	1	a2
v	0	b1
w	1	c2

ABCD

x	1	c1
y	1	d1
v	0	b2
w	1	c2

hidden stack

w	b2
x	a1

y	a2
w	b2
x	a1

Summary

- Review of Midterm Exam.
- Although the exam was over, please review your answers and what you don't know correctly or sufficiently.
- Notations, Recursion, Scope, and Bindings are important even after the midterm.