Course Title	Programming Language Theory (ITP20005, Oct. 23 2020)			Score	___ / 25
Section	**TF5** / TF6 (circle or underline yours)	Student ID	21701065	Name	Wonpyo Hong 홍원표

I hereby testify before God and men that all of the answers are my own, and I have taken the exam to the best of my ability without resorting to unethical conduct.

Caution: Answer sheet without Student's signature is not valid. Signature _____홍원표____ (write or type)

*** Answers written in Korean will not be marked. (Exam duration: 80 minutes)**
*** For students who are taking this by using your laptop, please use [File] >> [Make a copy] menu, and save it into your shared google directory for the PL class. (Like Quiz02)**
*** Open reference test but do not blind copy and paste. Write sentences based on your understanding.**
*** While taking the exam online, you must turn on either webcam 'or' mic. This is just for symbolic action that we are taking the exam honestly together.**

1. For abstract syntax (grammar) of the AE language, we can define a data structure (type). Here are two different types for the abstract syntax.

Type 1	Type 2
(define-type **AE** [**num** (n number?)] [**add** (lhs **num**?) (rhs **num**?)] [**sub** (lhs **num**?) (rhs **num**?)])	(define-type **AE** [**num** (n number?)] [**add** (lhs **AE**?) (rhs **AE**?)] [**sub** (lhs **AE**?) (rhs **AE**?)])

(1) Provide an example expression in abstract syntax that is valid for both **Type 1** and **Type 2** (1P)

(add (num 1) (num 2))

(2) Provide an example expression in abstract syntax that is valid only for **Type 2** (1P)

(add (add (num 1) (num 2)) (num 3))

(3) Why should we use the type **AE** for the lhs and rhs sub-expressions (**Type 2**) rather than the type **num (Type 1)**? (1P)

The sub expression of only Type 2 can be any AE expression.

2. Assume that we updated FAE into MFAE that supports multiple arguments (one or multiple function arguments). Evaluate the following expressions:

(1) {with {y 3} {{fun {x} {+ x y}} {+ 27 5 }} **(2P)**
(* Put your age and the month digit(s) of your birthday in ___ and ___ respectively. And then, evaluate.)

(interp (parse '{fun {x} { + x y}} 33} [x = 33, y = 3]
36

(2) {with {z _5____ } {{fun {x y} {- z {+ x y}}} 2 3}} **(1P)**
(* Put the last digit of your student id in _____ and then evaluate.)

{fun { x y } { - 5 { + x y } } } 2 3 } } [z=5 x = 2, y = 3]
0

(3) {fun {x y z} {+ x y z} 2 3} **(1P)**
(Hint. Intentionally provided two parameters for the function having three arguments.)

error

(4) Explain briefly why you answered like in (3). **(2P)**

One argument will not be provided with the parameter, thus it will be free identifier.

3. Write a BNF for the MFAE that supports multiple arguments from Problem 2. (2P)

<MFAE> ::= <num>
| { + <MFAE> <MFAE> }
| { - <MFAE> <MFAE> }
| <id MFAE>
| { <fun {<id MFAE> <MFAE> }
| { <MFAE> <MFAE> }

4. Explain why you design the BNF like in Problem 3. If you think your design is not perfect, discuss your problem. If you can explain issues in your design correctly, you can still get 2 points even though your BNF design has errors. (2P)

The main difference between MFAE and FAE is that MFAE should suffice multiple variables (meaning multiple arguments). Therefore, the number of variables shown on left handside of '{ <fun {<id MFAE> <MFAE> }' must match with the right hand side number. In order to do this, <id> has been replaced with <id MFAE> so that it can recursively find the total number of ids.

5. Discuss why substitution is an important principle in programming languages. (2P)

Substitution is an important principle because it reduces mistakes that humans may make like typical syntax errors or simple typing-mistakes. Also, it reduces reduntant computations.

6. In your preferred programming (c, java, python, etc...), what error message is printed out when you have a free identifier while interpreting or compiling your program? (2P)

Your preferred language:

C

Error messages for a free identifier in your preferred language:

Identifier not found

7. Find or guess any syntactic sugar in your preferred language such as C, Java, python, etc., and then explain why you think it as syntactic sugar. (Do not provide examples discussed in the classroom, i.e. the multiplication syntax by the add syntax or the 'with' syntax by the 'app' syntax. All other examples are fine. What you answered may not be a real syntactic sugar. But if your reasoning is acceptable, you can get two points. The purpose of this problem is to check your understanding about syntactic sugar.) **(2P)**

In C, there is something called switch statement, which is a syntactic sugar. Syntactic sugar is basically an another way that consolidate programming tasks in fewer characters (or just other characters). For example, we could write multiple if statements, starting the first one with if and the rest of them with else ifs. However, we could also have a switch function with multiple cases in it which works just as the multiple if statements.

8. How do you want to desugar for the syntactic sugar you answered in Problem 7. (2P)

The way to desugar this expression is basically converting it back to the way that parsers can understand, which is the multiple ifs.

9. We are designing a new updated language, SDFAE, that supports both static and dynamic scope for your functions.

(1) Write a BNF for SDFAE. (Update from the BNF of FAE.) (Hint: Add syntax that allows different kinds of functions. One function is for explicitly supporting dynamic scope and another function is for explicitly supporting static scope. You can choose proper keywords for your syntax.) **(1P)**

```
<SDFAE> ::= <num>
          | { + <SDFAE> <SDFAE> }
          | { - <SDFAE> <SDFAE> }
          | <id>
          | { <fun {<id> <SDFAE> }
          | { <SDFAE> <SDFAE> }
```

(2) Write two example expressions from SDFAE and evaluate them. One example must show static scope works clearly and another example expression must show dynamic scope works clearly (hint: Based on your SDFAE BNF, you can modify the example expression that suffers from a dynamic scope issue from L12 Slide 58). **(1P)**

(interp (parse '{with {x 3} {with {f {fun {y} {+ x y}}} {with {x 5} {f 4}}}}}) (mtSub))

10. Compare deferred substitution and laziness in various aspects. (2P)

Deferred substitution is the way of substituting values of variables at latest step. The simple difference of this and laziness is that deferred does not compute the result until the caller actually uses the value. However, laziness delay the whole computation until the result is required.

- END -

In their hearts humans plan their course, but the Lord establishes their steps. (Proverbs 16:9)