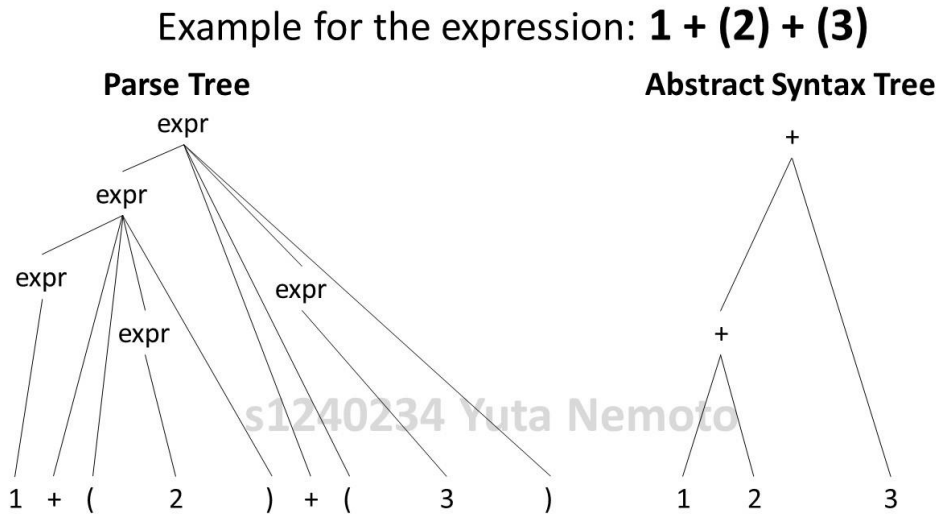
Quiz 8

1. What are the essential issues of semantic analysis?
    - Abstract Syntax Trees (AST)
    - Scope
    - Symbol tables
    - Type checking

2. Give an example to show the difference between AST and parse tree?



3. Briefly explain the AST construction?
    - **AST Nodes constructed during parsing**
    - **Bottom-up parser**
        - Grammar rules annotated with actions for AST construction
        - When node is constructed all children available (already constructed)
    - **Top-down parser**
        - More complicated

For example, the expression 1 + (2) + (3) is recognized as follows

1 + (2) + (3)

expr + ((2) + (3)

expr + (expr) + (3)

expr + (3)

expr + (expr)

expr

Then it can constructs the parse tree and can be more abstractive as follows

expr : expr '+' expr { $$ = plus($1, $3);} | '(' expr ')' {$$ = $2;} | INT_CONST {$$ = int_const($1);}

4. What are the scope rules? And briefly explain the scope of an identifier?

Scope rules means the definition of identifiers in the program. In one program, there're no multiple definition of the same identifier. Local variables are defined before it's used. And program exactly confirms to scope rules.

And about the scope of an identifier, for example, the scope of a variable declaration is the part of the program where that variable is visible.

5. Explain about the scope definition in C and Java programming languages?

In C language program,

- Global scope holds variables and functions
- No function nesting
- Block level scope introduces variables and labels
- File level scope with static variables that are not visible outside the file (global otherwise)

In Java language program,

- Limited global name space with only public classes
- Fields and methods in a public class can be public (visible to classes in other packages)
- Fields and methods in a class are visible to all classes in the same package unless declared as private
- Class variables visible to all objects of the same class.

6. Briefly explain the types of scoping?

- **Static Scoping**

  Scope of a variable determined from the source code.

  The rule is "Most Closely Nested", and it's used by most languages.

- **Dynamic Scoping**

  In this scoping, current call tree determines the relevant declaration of a variable use.

7. Give an example to show the difference between static and dynamic scoping?

For example, for the code below:

```
// Global variable
int x = 10;

// Function f called by g()
int f() { return x }

// Function g() called by main process.
int g() {
        int x = 5;
        return f();
}

// Main process
int main() {
        printf("%d\n", g() );
        return 0;
}
```

In the case of **Static Scope**, because of the Most Closely Nested Rule means the scope of a particular declaration is given by the most closely nested rule, the function f returns **x = 10** which is initialized as global variable.

On the other hand, in the case of **Dynamic Scope**, the function f returns **x = 5** because in the specific function, it can references the local variables and all visible variables in all active subprograms. In this case, the process of function g() is not terminated yet while the execution of function f(), so at the return statement in the function f(), it references the value of x in the function g() which calls the function f().