Quiz 1

1. What are the issues that drive the compiler design?

- Correctness
- Runtime and Compile time Speed
- Space
- Feedback to user
- Debugging

2. Explain about the difference between compilers and interpreters?

The difference between them is just there's a process of optimization and translation into the machine code or not. As the features of Compilers, the source code written in programming language such as Java, C or C++ is translated into machine code when we execute the compilation, so it doesn't need to interpret the structure of programming code when it runs in the computer. It means compiled program is much faster than the program with interpreters. And also, the compiler makes it possible to generate the machine code optimized for each operating system. For example, the code written in Java can be machine code in both of Mac OS and Windows through the compiler. On the other hand, with the Interpreters, the read input program is directly executed at runtime. It means we don't need the compilation process, but the runtime can be slower than the compiled program.

3. What are the objectives of the compilers?

- Fundamental principles
  - Compilers shall preserve the meaning of the input program correctly.
  - Compilers shall do something of value.
- Evaluation criteria for quality of a compiler
  - The speed of the compiled code
  - The size of the space of the compiled code
  - The compiler provides feedback on incorrect program or not
  - Possibility of the debugging of incorrect program
  - The translation speed

4. What is the compiler structure?

The structure of the Compiler can be recognized as 3 processes: Front end, Mid end, Back end.

Front end: processes like scanning, parsing, context-sensitive analysis for understanding the source program.

Mid end(Optimizer): processes like Data-flow analysis, redundancy elimination, computation re-structuring for improving the input program.

Back end: processes like Instruction selection and scheduling, register allocation for generating executable code for target machine.

Each of these processes are connected by using **IR**: Intermediate (internal) representation of the input, such as Abstract syntax tree, symbol table, control-flow graph.

5. Explain about how to instruct compilers?

First, we need to write a program using a programming language such as Java, C, or C++. Then the compiler for each programming language translate the code we wrote into Assembly language the computer can understand. At the translation, compiler read and understand what's written in the program, precisely determine what actions it require, figure-out how to faithfully carry-out those actions, and instruct the computer to carry out those actions.

6. Explain about the input and output of compilers?

About the input to the compiler, it's a standard imperative language such as Java, C, or C++, including state like variables, structures and arrays, and computation like expressions, assignment statements, control flow and procedures.

About the output of the compiler, it's including state such as registers or memory with flat address space, machine code expressing load/store instructions, arithmetic/logical operations on registers or branch instructions.

In summerly, the input code is a program code written in high-level language, and the output code is machine language like assembly code.

7. Explain about compiler development test cycle?

First, we need to put the development compiler source code in resident compiler, then it makes development compiler. Next, we put the application source code in the development compiler, then it generates compiled application. Now, we can give an application input to the compiled application, then it gives us the application output. Finally, we can compare the application output and the expected output using output verifier. It shows the result is correct or not.

8. Write a program in Java that translate Infix expression to its postfix equivalent.

- How to execute

  First, compile the program using the command:

  javac Translator.java Determination.java InfixPostfixTranslator.java

  Then, when you execute the program, you should give the expression as an argument like below.

  java InfixPostfixTranslator 2+3*2

- Program source code

  [Translator.java]

```
package lecture01;

public abstract class Translator {
        public Translator() {}
        public abstract String translate(String str);
}
```

[Determination.java]

```
package lecture01;

public final class Determination {
	public static final int getPriority(char c) {
		switch(c) {
		case '+':
		case '-':
			return 1;
		case '*':
		case '/':
			return 2;
		case '^':
			return 3;
		default:
			return 0;
		}
	}
}
```

[InfixPostfixTranslator.java]

```
package lecture01;

import java.util.Stack;

public class InfixPostfixTranslator extends Translator {

	/* Translate function */
	@Override
	public String translate(String str) {
		// String to return the translated result.
		String result = "";
		// Prepare stack to process the translation.
		Stack<Character> stack = new Stack<>();

		for(int i = 0; i<str.length(); ++i) {
			// Scan a character from the given string.
			char c = str.charAt(i);
			// If the scanned character is an operand, add it to output.
			if(Character.isLetterOrDigit(c)) result += c;
			// If the scanned character is '(', push it to the stack.
			else if(c == '(') stack.push(c);
```

```
                        // If the scanned character is ')', pop and output from the
    stack until '(' is encountered.
                        else if(c == ')') {
                                while(!stack.isEmpty() && stack.peek() != '(') result +=
    stack.pop();
                                if(!stack.isEmpty() && stack.peek() != '(') return
    "Invalid expression";
                                else stack.pop();
                        }
                        // If the scanned character is an operator.
                        else {
                                while(!stack.isEmpty() && Determination.getPriority(c)
    <= Determination.getPriority(stack.peek())) result += stack.pop();
                                stack.push(c);
                        }
                }
                // Pop all the operators from the stack.
                while(!stack.isEmpty()) result += stack.pop();
                return result;
        }

        /* Main Process */
        public static void main(String[] args) {
                InfixPostfixTranslator translator = new InfixPostfixTranslator();
                try {
                        System.out.println(translator.translate(args[0]));
                } catch(ArrayIndexOutOfBoundsException e) {
                        System.out.println("Please enter the expression as an
    argument.");
                }
        }

    }
```

- Example result
  - Input: 2+3*4 Output: 234*+
  - Input: 3/5+2*1 Output: 35/21*+
  - Input: a+b*10 Output: ab10*+