**Quiz 7**

1. **Explain the general approach for writing LR parser?**

The construction is done automatically by a tool such as the Unix program yacc.

1. **Briefly explain about Yacc?**

Yacc is Yet Another Compiler-Compiler

It automatically generate LALR parsers.

It’s created by S.C. Johnson in 1970’s.

1. **How the Yacc tool is used to generate parsers?**

First, the Yacc source program is compiled with Yacc compiler, and generates y.tab.c program. Next, the y.tab.c program is compiled with C compiler and it generates the executable file a.out: Parser. It works like generating parse tree from the input tokens.

1. **Explain the structure of the Yacc program?**

The yacc program is mainly divided in 3 parts: Definition section, Production rules section and C auxiliary subroutines. And those are divided by the marker “%%”.

**Definition section**:

It contains token declarations. Tokens are recognized in lexer.

**Production rules section**:

It defines how to “understand” the input language, and what actions to take for each “sentence”.

**C auxiliary subroutines**:

Any user code. For example, a main function to call the parser function yyparse().

1. **Explain how to run a Yacc program?**

The executing command is below:

**yacc –d –v my\_prog(the name of the yacc program).y**

**gcc –o y.tab.c –ly**

-d option creates a file “y.tab.h”, which contains a #define statement for each terminal declared.

Place #include “y.tab.h” in between the %{ and %} to use the tokens in the functions section.

-v option creates a file “y.output”, which contains useful information on debugging.

1. **Explain with example about the production rules section in the Yacc program?**

For a production rule: nontermsym → symbol1 symbol2 … | symbol3 symbol4 … | …

The Yacc program is written like:

nontermsym : symbol1 symbol2 … { actions }

|symbol3 symbol4 … { actions }

|…

;

So, for example, for a production rule: expr → expr + expr

The Yacc program is like: expr : expr ‘+’ expr {$$ = $1 + $3}

Here, $$ is the value of non-terminal on lhs, and values like $3 is one of n-th symbol on rhs.

1. **Briefly explain about the Yacc errors?**

Yacc can not accept ambiguous grammars, nor can it accept grammars requiring two or more symbols of lookahead. The two most common error messages are **shift-reduce conflict**: where the parser would have a choice as to whether it shifts the next symbol from the input, or reduces the current symbols on the top of the stack, and **reduce-reduce conflict**: where the parser has a choice of rules to reduce the stack.

For example, Yacc code:

Expr : INT\_T | Expr + Expr ;

Causes a shift-reduce error, because INT\_T + INT\_T + INT\_T can be parsed in two ways.

Another example, Yacc code:

Animal : Dog | Cat ;

Dog : FRED\_T ;

Cat : FRED\_T ;

Causes a reduce-reduce error, because FRED\_T can be parsed in two ways.