Compiler Construction

YACC

Mr. Usman Wajid

usman.wajid@nu.edu.pk

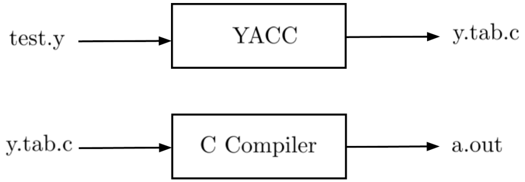
National University of Computer & Emerging Sciences

What is YACC?

YACC

YACC - Yet Another Compiler Compiler - is a tool for construction of automatic LALR parser generator.

- YACC code is specified extension .y
  For example, **test.y**
- Run this file with the YACC command as:
  $yacc test.y
- This produces **y.tab.c**
- compile ya.tab.c with a C-compiler as:
  $gcc y.tab.c
- This produces **a.out**
- execute a.out using thecommand:
  $/a.out



Preparing the YACC Specification File

- YACC specification file consists of three sections separated by %% as follows:

  1. declarations

     %%

  2. Translation rules

     %%

  3. subroutines

YACC Declaration Section

It contains two types of declarations

1. **C-declarations**
2. **YACC declarations**

YACC C-declaration

C-declaration

C-declarations are enclosed with %{ and %}

- Here we can write:
    - header files such as #include <stdio.h>, #include <stdlib.h> etc
    - global variables such as int result = 0;
- This may be used for defining subroutines or action part of grammar rules.

```c
%{
/* C Declarations */
#include <stdio.h>
#include <stdlib.h>

int yylex(void);
void yyerror(const char *s);

/* Global variable */
int result = 0;
%}
```

YACC YACC-declaration

YACC-declarations

YACC declarations are nothing but tokens or terminals

- For example, a ”num” token can be defined as: %token num
- the **precedence** and **associativity**:
  - %left and %right means left and right associativity respectively
  - tokens on the same line means same precedence
  - lines declared below have higher precedence than the lines declared above

```
%{
#include <stdio.h>
#include <stdlib.h>

int yylex(void);
void yyerror(const char *s);

int result = 0;
%}

/* YACC Declarations */
%token NUMBER

%left '+' '-'
%left '*' '/'

%%
```

YACC Translation Rules

- In translation rules section, grammar is written with associated actions
- An action is defined with a set of C statements.
- $$ refers to the attribute of the current symbol andsingle $ refers to stored values in other symbols attributes
  - For example, A: B C D { $$ = $1 + $2 + $3; }
    then$$ has the attribute value to be computed for A. Whereas, $1, $2 and $3 has the attribute values of symbols B, C and D respectively
  - In another example, E : '(' E ')' { $$ = $2;}
    Here, $$ refers to the attribute of E on left hand side of the production and $2 refers to the stored attribute value of E on the right hand side of the production

YACC Translation Rules continued

```
/* Grammar Rules */
expr:
expr '+' expr   { $$ = $1 + $3; result = $$; }
| expr '-' expr   { $$ = $1 - $3; result = $$; }
| expr '*' expr   { $$ = $1 * $3; result = $$; }
| expr '/' expr   {
        if ($3 == 0) {
                yyerror("Division by zero");
                exit(1);
        }
        $$ = $1 / $3;
        result = $$;
}
| '(' expr ')'      { $$ = $2; result = $$; }
| NUMBER            { $$ = $1; result = $$; }
;

%%
```

YACC subroutines section

```
void yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
}

int main(void) {
        printf("Enter an arithmetic expression:\n");
        if (yyparse() == 0) {
                printf("Final result = %d\n", result);
        }
return 0;
}
```

Lexical analysis for YACC

- The lexical analysis for YACC specification is done by using the yylex() subroutine
- The yylex() subroutine is only declared in the YACC specification file but not defined.
- Therefore, it is defined by two ways:
  1. Define it in the subroutine section by the whole lexical analyzer code in C language in the sub routine section of YACC specification file
  2. Or import it from the Flex program

yylex() code for YACC specification file using Flex proram

```
%{
#include "y.tab.h"   // Required for token definitions like NUMBER
#include <stdlib.h>
%}

%%

[0-9]+        { yylval = atoi(yytext); return NUMBER; }
[ \t\n]+      ; // Ignore whitespace
[+\-*/()]     { return yytext[0]; } // Return single-character tokens

%%

int yywrap(void) {
        return 1;
}
```

Finally, How to Perform translation?

- Open PowerShell prompt
- Go to the location where expr.y and lexer.l files are stored.
- Then write the following commands in sequence:

1. #yacc -d expr.y

   Generates y.tab.c and y.tab.h (to be used by the lexer.l file)

2. #lex lexer.l

   Generates lex.yy.c

3. #gcc lex.yy.c y.tab.c

   Compile and creates a.out.exe file

4. #./a.out

   Runs the parser