Experiment 9

Aim: Implementation of Calculator using LEX and YACC

Description

Implement an expression evaluator which supports integer and decimal numbers. The following operators must be supported, +,-,*,/,(,).

To remember when using lex and yacc together

In the declaration section of lex program, include prog_name.tab.h for making lexer to read the input symbol, so that it can send tokens to Yacc parser. Also use -d switch while invoking bison.

Recognize a valid arithmetic expression

The lexical analyzer part should do the following:

- When a number is encountered in the input, return the token NUMBER to the parser and it's value should be used to set yylval.
- if input contains a tab space, ignore it.
- any other characters should be returned as such.
- if newline is encountered, it means end of expression, so return 0 to terminate lexer execution.

```
%%
([1-9][0-9]+|[0-9])?(\.[0-9]+)? {
yylval.val=strtod(yytext,NULL);
return NUMBER;  }
[ \t] ;
[\n] return 0;
. return yytext[0];
%%
```

Calculator using LEX and YACC

For the YACC part, we use the following grammar,
$E \rightarrow E + E | E * E | E - E | E / E | (E) | NUMBER$

- The terminal symbol$NUMBER$ and operators are the tokens recognized by lexical analyzer.
- The precedence and associativity of operators has to be specified in the YACC specification.
- First define tokens which are getting returned from lexer, in the declaration section
- Then assign precedence and associativity of operators, first defined ones will have least preference. left means left-associativity, and right means right-associativity. , also in the declaration section
- The grammar productions are to be written in the rules section

Things to note

- Here, we have to recognize decimal numbers.
- Matching the pattern in our lexer is easy.
- But assigning the value of the number to yylval is not straight forward as its default value will be integer.
- we have to define %union as float val.
- Also in the rules section we have to write $<val>$ = $<val>1+$<val>3 instead of $$=$$1+$$3 as in case of integers.
- So the rule corresponding to the production $E \rightarrow E + E$ is $<val>$ = $<val>1+$<val>3
- We can print invalid expression in yyerror() function

Remember

Use the -d switch, while invoking bison to generate prog_name.tab.c

YACC code sample

```
%{
/* Definition section */
#include<stdio.h>
%}
/* The %union declaration modifies the type of yylval, whic
%union {
float val;
}
%token NUMBER
%left ’+’ ’-’
%left ’*’ ’/’
%left ’(’ ’)’
```

YACC code sample

```
/* Rule Section */
%%
ArithmeticExpression: E{ printf("\nResult=%g\n", $<val>$);
return 0;
};
E:E’+’E {$<val>$=$<val>1+$<val>3;}
|E’-’E {$<val>$=$<val>1-$<val>3;}
|E’*’E {$<val>$=$<val>1*$<val>3;}
|E’/’E {$<val>$=$<val>1/$<val>3;}
|’(’E’)’ {$<val>$=$<val>2;}
| NUMBER {$<val>$=$<val>1;}
;
%%
```

YACC code sample

```
%{
/* Definition section */
#include<stdio.h>
%}
/* The %union declaration modifies the type of yylval, whic
%union {
float val;
}
%token NUMBER
%left ’+’ ’-’
%left ’*’ ’/’
%left ’(’ ’)’
```

YACC code sample

```
//auxiliary functions
void main()
{
printf("\nEnter any expression which can have operators
    +,-,*,/ and parantheses:\n");
yyparse();
}
void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n"
   );
}
```

Header files in lex part

The header files stdio.h, stdlib.h and calc.tab.h(generated by using -d switch with bison) must be included in the lex file.