Compiler Design

YACC (Yet Another Compiler Compiler) Programming

Installation guidelines of flex and bison in windows.

1. First, install flex and bison. Use your preferred directory without spaces in the name. I suggest C:\GnuWin32. Do not install it in the default C:\Program Files (x86)\GnuWin32, because bison has problems with spaces in directory names.
2. Also install Dev-CPP in the default directory (C:\Dev-Cpp).
3. After that, set the PATH variable. To do that, copy C:\Dev-Cpp\bin;C:\GnuWin32\bin and append it to the end of the PATH variable.
4. Now it is installed.

How to execute YACC(in command prompt).

Suppose your *lex* and *yacc* file name is *test.l* and *test.y* respectively.

```
yacc –d test.y –y          //Compilation of yacc file. It will generate y.tab.c and y.tab.h
lex test.l                 //Compilation of lex file. It will generate lex.yy.c
gcc y.tab.c lex.yy.c       //Compilation of y.tab.c and lex.yy.c
a                          //to execute
```

Few programs:

1. Program to recognize strings Generated by the Grammar $G = \{ S \rightarrow AA, A \rightarrow aA \mid b \}$.

YACC part (Prog1.y):

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

%start Stmt
%token ID1 ID2 NL
%%              /* beginning of rules section */

Stmt: S NL {printf("\nSuccessful parsing."); exit(0);}
        ;
S: A A {printf("\nReduced by production 1.");}
  ;
A: ID1 A {printf("\nReduced by production 2.");}
   |
     ID2 {printf("\nReduced by production 3.");}
   ;

%%
```

```
int main(){
       printf("\nGive a string:");
       yyparse();
}
yyerror(char *s){
        printf("\nError: %s\n",s);
}

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

LEX part (Prog1.l):

```
%{
#include "y.tab.h"
%}
%%
[a] { printf("\nReceived 'a'."); return (ID1);}
[b] { printf("\nReceived 'b'."); return (ID2);}
\n { return NL ;}
. { return yytext[0]; }
%%
```

Output 1:

```
Give a string: aabb

Received 'a'.
Received 'a'.
Received 'b'.
Reduced by production 3.
Reduced by production 2.
Reduced by production 2.
Received 'b'.
Reduced by production 3.
Reduced by production 1.
Successful parsing.
```

Output 2:

```
Give a string: aab

Received 'a'.
Received 'a'.
Received 'b'.
Reduced by production 3.
Reduced by production 2.
Reduced by production 2.
Error: syntax error
```

2. Program to recognize strings Generated by the language $L = \{a^n b^n, n \geq 0\}$.

The corresponding grammar of the language is $G = \{\ S \rightarrow ASB \mid \varepsilon,\ \ A \rightarrow a,\ \ B \rightarrow b\ \ \}$.

YACC part (Prog2.y):

```
%{
#include<stdio.h>
%}
%start Stmt
%token NL
%%                    /* beginning of rules section */
Stmt: S NL {printf("\nSuccessful parsing."); exit(0);}
 ;
S: A S B
  |
 ;
A: 'a'
 ;
B: 'b'
 ;
%%
int main(){
    printf("\nGive a string:");
    yyparse();
}

yyerror(char *s){
      printf("\nError: %s\n",s);
}

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

LEX part (Prog2.l):

```
%{
#include "y.tab.h"
%}
%%
\n { return NL ;}
. { return yytext[0]; }
%%
```

Output:

```
Give a string: aaabbb
Successful parsing.
```

3. **Program to evaluate the simple mathematical expression of the following grammar. The value of *a* will be given as numeric.**

$$T \rightarrow T+T \mid T-T \mid T*T \mid T/T \mid (T) \mid a$$

YACC part (Prog3.y):

```
%{
#include<stdio.h>
%}
%start Stmt
%token ID NL
%left '+' '-'
%left '*' '/'
%%
 /* $$ indicates attribute value of the head symbol, i.e., LHS of the production. $1 indicates
attribute value of the 1st symbol of RHS of the production. Therefore, here, $2 indicates '+'
symbol for first production. */
Stmt: T NL {printf("\nThe evaluated value is:%f",(float)$1); exit(0);}
 ;
T: T '+' T {$$ = $1+$3;}
   |T '-' T {$$ = $1-$3;}
   |T '*' T {$$ = $1*$3;}
   |T '/ ' T {  if($3==0)
            {
              printf("Cannot divide by 0");
              exit(0);
            }
            else
              $$ =$1/$3;
          }
   |'('T')' {$$ = $2;}
   |ID {$$ = $1;}
 ;
%%
int main(){
 printf("\nGive a numeric expression:");
 yyparse();
}

yyerror(char *s){
    printf("\nError: %s\n",s);
}

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

LEX part (Prog3.l):

```
%{
#include "y.tab.h"
#include<stdlib.h>
%}
%%
\n {return NL;}
[0-9]+ {yylval= atoi(yytext); return ID;}
[0-9]+"."[0-9]+ {yylval= atof(yytext); return ID;}
. {return yytext[0];}
%%
```

Output:

```
Give a numeric expression: 45.24+66.00/3
The evaluated value is: 67.000000
```