Saivya Singh
220905730
CSE D
44

Lab 10 : Introduction to Bison

Q1 . *To check a valid declaration statement.*

Code :

Bison

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(char *msg);
%}

/* Token definitions */
%token INT FLOAT CHAR ID NUM SEMI COMMA ASSIGN

%%
decl_stmt: type_specifier decl_list SEMI
          { printf("Valid Declaration\n"); exit(0); }
        ;

type_specifier: INT
             | FLOAT
             | CHAR
             ;

decl_list: decl
        | decl_list COMMA decl
        ;

decl: ID
    | ID ASSIGN NUM
    ;
%%

int yyerror(char *msg)
{
    printf("Invalid Declaration\n");
    exit(0);
}

int main(void)
{
    printf("Enter a declaration statement:\n");
```

```
    yyparse();
    return 0;
}
```

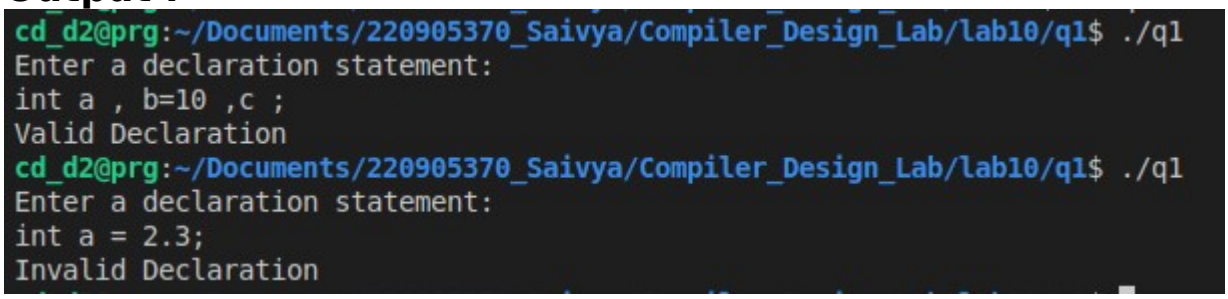
Flex

```
%{
#include "y.tab.h"
#include <stdlib.h>
%}

%%
[0-9]+          { yylval = atoi(yytext); return NUMBER; }
\n              { return '\n'; }
"n"             { return 'n'; }
"+"             { return '+'; }
"-"             { return '-'; }
"*"             { return '*'; }
"/"             { return '/'; }
"^"             { return '^'; }
[ \t]+          { /* skip whitespace */ }
.               { return yytext[0]; }
%%

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

Output :



Q2. *To check a valid decision making statements.*

Code :

Bison

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(char *msg);
%}

%token IF ELSE LPAREN RPAREN LBRACE RBRACE ID NUM GT ASSIGN SEMI
%left GT

%%
program:
      stmt { printf("Valid decision statement\n"); exit(0); }
    ;
stmt:
      matched_stmt
    | unmatched_stmt
    ;
matched_stmt:
      IF LPAREN expr RPAREN matched_stmt ELSE matched_stmt
    | other_stmt
    ;
unmatched_stmt:
      IF LPAREN expr RPAREN stmt
    ;
other_stmt:
      block
    | assign_stmt
    ;
block:
      LBRACE stmt_list RBRACE
    ;
stmt_list:

    | stmt_list stmt
    ;
assign_stmt:
      ID ASSIGN expr SEMI
    ;
expr:
      expr GT expr
    | ID
    | NUM
    ;
%%
```

```
int yyerror(char *msg)
{
   printf("Invalid decision statement\n");
   exit(0);
}

int main(void)
{
   printf("Enter a decision making statement:\n");
   yyparse();
   return 0;
}
```

Flex

```
%{
#include "y.tab.h"
%}

%%
"if"          { return IF; }
"else"         { return ELSE; }
"("           { return LPAREN; }
")"           { return RPAREN; }
"{"           { return LBRACE; }
"}"           { return RBRACE; }
";"           { return SEMI; }
">"           { return GT; }
"="           { return ASSIGN; }
[0-9]+         { return NUM; }
[a-zA-Z][a-zA-Z0-9]* { return ID; }
[ \t\n]+       { /* Skip whitespace */ }
.            { return yytext[0]; }
%%

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

Output :

```
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab10/q2$ ./q2
Enter a decision making statement:
if (x > 0) x = 10; else x = 20;
Valid decision statement
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab10/q2$ ./q2
Enter a decision making statement:
if (x > 0 x = 10;
Invalid decision statement
```

Q3. *To evaluate an arithmetic expression involving operations +,-,* and /.*

Code :

Bison

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

%token NUMBER
%token LPAREN RPAREN
%left '+' '-'
%left '*' '/'
%left NEG

%%
input:
     /* empty */
   | input line
   ;
line:
     expr '\n' { printf("Result = %d\n", $1); }
   ;
expr:
     expr '+' expr { $$ = $1 + $3; }
   | expr '-' expr { $$ = $1 - $3; }
   | expr '*' expr { $$ = $1 * $3; }
   | expr '/' expr { if($3 == 0) { printf("Division by zero error\n"); exit(1); } else
$$ = $1 / $3; }
   | '-' expr %prec NEG { $$ = -$2; }
   | LPAREN expr RPAREN { $$ = $2; }
   | NUMBER { $$ = $1; }
   ;
%%

int main(void)
{
   printf("Enter arithmetic expression:\n");
   yyparse();
   return 0;
}

int yyerror(char *s)
{
   fprintf(stderr, "Error: %s\n", s);
   exit(1);
}
```

Flex

```
%{
#include "y.tab.h"
%}

%%
[0-9]+          { yylval = atoi(yytext); return NUMBER; }
"("             { return LPAREN; }
")"             { return RPAREN; }
"+"              { return '+'; }
"-"             { return '-'; }
"*"              { return '*'; }
"/"             { return '/'; }
\n              { return '\n'; }
[ \t]+           ;
.              { return yytext[0]; }
%%
int yywrap(void) { return 1; }
```

Output :

```
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab10/q3$ ./q3
Enter arithmetic expression:
9*7
Result = 63
9+6
Result = 15
^C
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab10/q3$ ./q3
Enter arithmetic expression:
9&2
Error: syntax error
cd_d2@prg:~/Documents/220905370_Saivya/Compiler_Design_Lab/lab10/q3$ ./q3
Enter arithmetic expression:
2*9+8/4
Result = 20
^C
```

Q4. *To validate a simple calculator using postfix notation. The grammar rules are as follows –*

input → input line | ε
line → ‘\n’ | exp ‘\n’
exp → num | exp exp ‘+’
| exp exp ‘-‘
| exp exp ‘’*
| exp exp ‘/’
| exp exp ‘^’
| exp ‘n’

Code :

Bison

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(char *msg);
%}

%token NUMBER

%%
input:
     /* empty */
   | input line
   ;
line:
     '\n'
   | exp '\n'
   ;
exp:
     NUMBER
   | exp exp '+'
   | exp exp '-'
   | exp exp '*'
   | exp exp '/'
   | exp exp '^'
   | exp 'n'
   ;
%%

int yyerror(char *msg)
{
   printf("Invalid postfix expression\n");
   exit(1);
}

int main(void)
```

```
{
    printf("Enter postfix expression:\n");
    yyparse();
    printf("Valid postfix expression\n");
    return 0;
}
```

Flex

```
%{
#include "y.tab.h"
#include <stdlib.h>
%}

%%
[0-9]+          { yylval = atoi(yytext); return NUMBER; }
\n              { return '\n'; }
"n"             { return 'n'; }
"+"             { return '+'; }
"-"             { return '-'; }
"*"             { return '*'; }
"/"             { return '/'; }
"^"             { return '^'; }
[ \t]+          { /* skip whitespace */ }
.               { return yytext[0]; }
%%

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

Output :