LAB 10 – INTRODUCTION TO BISON

Name: Pranamya G Kulal
Class: CSE A
Roll no: 8
Reg no: 220905018

Q1) To check a valid declaration statement.

i) q1.y

```
%{
	#include<stdio.h>
	#include<stdlib.h>
	int yylex();
	int yyerror();
%}
%token INT ID SC CM NL
%%
stmt: DC NL
	;
DC: DT IDL SC { printf("Valid declaration statement!\n"); exit(0);}
	;
DT: INT
	;
IDL: ID
	|ID CM IDL
	;

%%
int yyerror(char *msg) {
	printf("Invalid declaration statement!\n");
	exit(0);
}
void main () {
	printf("Enter the declaration statement:\n");
	yyparse();
}
```

ii) q1.l

```
%{
	#include "q1.tab.h"
%}
%%

"int" {return INT;}
";" {return SC;}
"," {return CM;}
[a-zA-Z][a-zA-Z0-9_]* {return ID; }
\n {return NL;}

%%

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

iii) Terminal output

```
Enter the declaration statement:
int a;
 Valid declaration statement!
```

Q2) To check a valid decision making statements.

i) q2.y

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

%token IF ELSE OB CB OP CP RELOP MULOP ADDOP ID NUM NL SC ASS

%%
statementList: statement statementList {printf("Vaild decision statement\n"); exit(0);}
                                        |
                                        ;
statement: assignStat SC
                    |decisionStat
                    ;
assignStat: ID ASS expn
                     ;
decisionStat: IF OP expn CP OB statementList CB dprime
                                       ;
dprime: ELSE OB statementList CB
            |
            ;
expn: simpleExpn eprime
          ;
eprime: RELOP simpleExpn
                   |
                   ;
simpleExpn:term seprime
                             ;
seprime: ADDOP term seprime
             |
             ;
term: factor tprime
          ;
tprime: MULOP factor tprime
                   |
                   ;
factor: ID
                   | NUM
                   ;
```

```
%%
int yyerror(char * msg){
	printf("Invalid decision statement!\n");
	return 1;
}
int main(){
	printf("Enter decision statement:\n");
	yyparse();
}
```

ii) q2.l

```
%{
	#include "q2.tab.h"
%}
%%

"if" {return IF;}
"else" {return ELSE;}
";" {return SC;}
"{" {return OB;}
"}" {return CB;}
"(" {return OP;}
")" {return CP;}
">="|"<="|"!="|"==" {return RELOP;}
"<"|">" {return RELOP;}
"=" {return ASS;}
"*"|"/"|"%" {return MULOP;}
"+"|"-" {return ADDOP;}
"\n" {return NL;}
[0-9]+ {return NUM;}
[A-Za-z_]+[A-Za-z_0-9]* {return ID;}

%%

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

iii) Terminal output

```
Enter decision statement:
if(a>b){}else{}
Vaild decision statement
```

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

i) q3.y

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

int yylex();
void yyerror(const char *s);
%}
```

```
%token NUM ADD SUB MUL DIV LPAREN RPAREN EOL

/* Define operator precedence and associativity */
%left ADD SUB   /* + and - have lower precedence */
%left MUL DIV   /* * and / have higher precedence */

%%

input: /* empty */
     | input line
     ;

line: EOL
    | exp EOL       { printf("Result: %d\n", $1); }
    ;

exp: NUM            { $$ = $1; }
   | exp ADD exp    { $$ = $1 + $3; }
   | exp SUB exp    { $$ = $1 - $3; }
   | exp MUL exp    { $$ = $1 * $3; }
   | exp DIV exp    {
         if ($3 == 0) {
             yyerror("Division by zero");
             $$ = 0;
         } else {
             $$ = $1 / $3;
         }
     }
   | LPAREN exp RPAREN { $$ = $2; }
   ;

%%

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

int main() {
    printf("Enter arithmetic expressions (e.g., 3 + 5 * 2):\n");
    yyparse();
    return 0;
}
```

ii) q3.l

```
%{
#include "q3.tab.h"
#include <stdlib.h>
%}
%%
[0-9]+ { yylval = atoi(yytext); return NUM; }
```

```
"+" { return ADD; }
"-" { return SUB; }
"*" { return MUL; }
"/" { return DIV; }
"(" { return LPAREN; }
")" { return RPAREN; }
[ \t] ; /* Ignore whitespace */
\n { return EOL; }
. { printf("Invalid character: %s\n", yytext); }
%%
int yywrap() {
 return 1;
}
```

iii) Terminal output

```
Enter arithmetic expressions (e.g., 3 + 5 * 2):
8 * 9 + 6
Result: 78
```

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’

i) q4.y

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

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

%union {
    int num;
}

%token <num> NUM
%type <num> exp

%start input

%%

input:
      input line
    | /* empty */
    ;

line:
      '\n'
    | exp '\n' { printf("Result: %d\n", $1); }
    ;

exp:
      NUM { $$ = $1; }
    | exp exp '+' { $$ = $1 + $2; }
    | exp exp '-' { $$ = $1 - $2; }
    | exp exp '*' { $$ = $1 * $2; }
    | exp exp '/' { $$ = $1 / $2; }
    | exp exp '^' { $$ = pow($1, $2); }
    | exp 'n' { $$ = -$1; }
    ;

%%

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

int main() {
    printf("Enter expressions in postfix notation (Ctrl+D to exit):\n");
    yyparse();
    return 0;
}
```

ii) q4.l

```
%{
#include "q4.tab.h"
%}
%%
[0-9]+ { yylval.num = atoi(yytext); return NUM; }
[\n] { return yytext[0]; }
[+\-*/^n] { return yytext[0]; }
[ \t] { /* ignore whitespace */ }
. { printf("Unexpected character: %s\n", yytext); }
%%
int yywrap(void) {
 return 1;
}
```

iii) Terminal output

```
Enter expressions in postfix notation (Ctrl+D to exit):
2 3 +
Result: 5
```