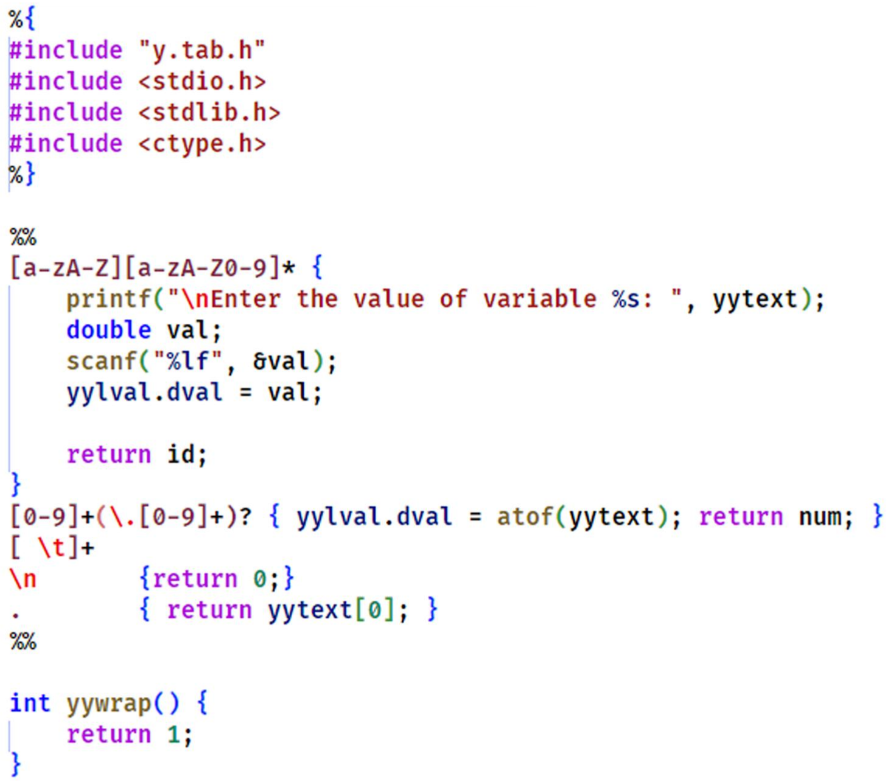
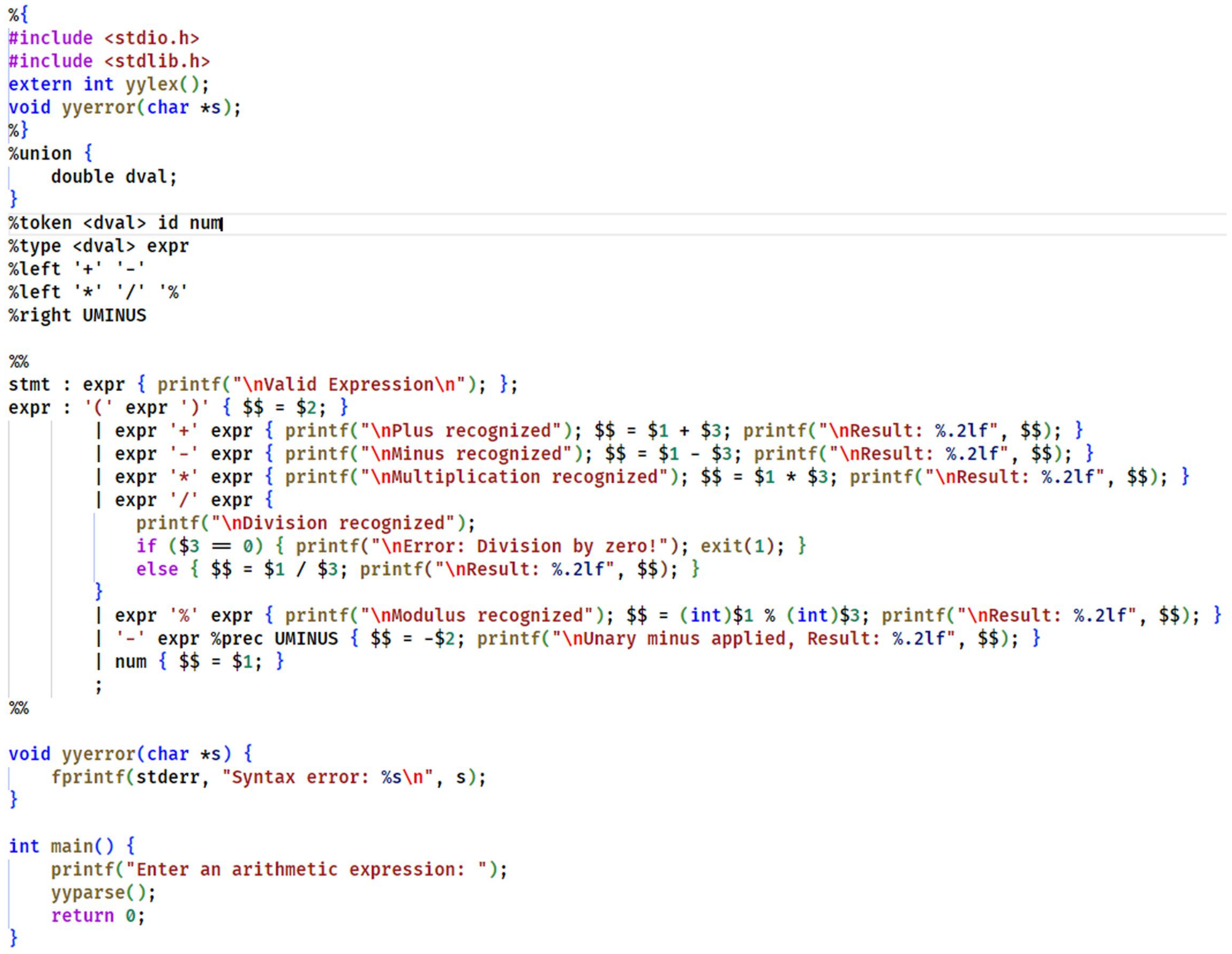
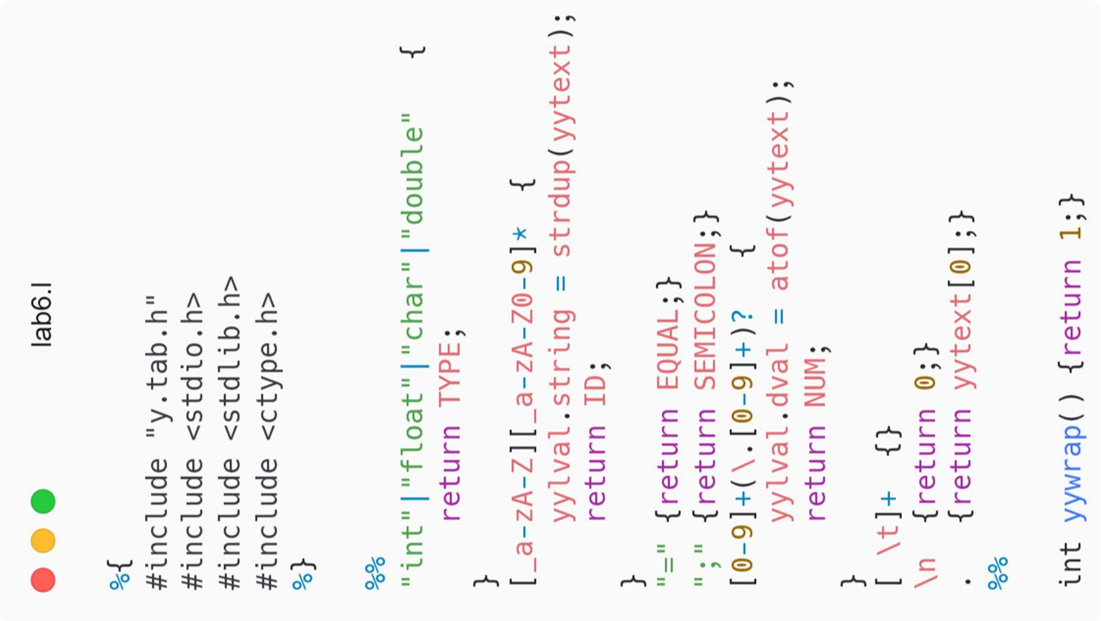
```
%{
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int operator_count = 0, operand_count = 0, top = -1, operand_top = -1, valid =
1;
char operator_stack[100];
int operand_stack[100];
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '/' || op == '*') return 2;
    return 0;
}
int calculate() {
    char op = operator_stack[top--];
    int b = operand_stack[operand_top--];
    int a = operand_stack[operand_top--];
    int res;
    switch(op) {
        case '+': res = a + b; break;
        case '-': res = a - b; break;
        case '*': res = a * b; break;
        case '/':
            if (b == 0) {
                printf("ERROR: Divide by zero!\n");
                valid = 0;
                return 0;
            }
            res = a / b;
            break;
    }
    operand_stack[++operand_top] = res;
}
%}

%%
"(" { operator_stack[++top] = '('; }
")" { while (top > -1 && operator_stack[top] != '(') { calculate(); } if (top > -1) top--; else { valid = 0; return
0; } }
[0-9]+ { operand_stack[++operand_top] = atoi(yytext); operand_count++; }
[+-/*] { while (top > -1 && precedence(operator_stack[top]) >= precedence(yytext[0])) { calculate(); }
operator_stack[++top] = yytext[0]; operator_count++; }
[ \t] ;
\n { while (top > -1) { calculate(); } return 0; }
%%

int main(void) {
yylex();
else if (operand_count > operator_count + 1) printf("Too many operands\n");
else if (operand_count <= operator_count) printf("Too many operators\n");
else if (operand_top > -1) printf("Result: %d\n", operand_stack[operand_top]);
return 0;
}
```



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

%%
[a-zA-Z][a-zA-Z0-9]* {
    printf("\nEnter the value of variable %s: ", yytext);
    double val;
    scanf("%lf", &val);
    yylval.dval = val;

    return id;
}
[0-9]+(\.[0-9]+)? { yylval.dval = atof(yytext); return num; }
[ \t]+
\n       {return 0;}
.        { return yytext[0]; }
%%

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

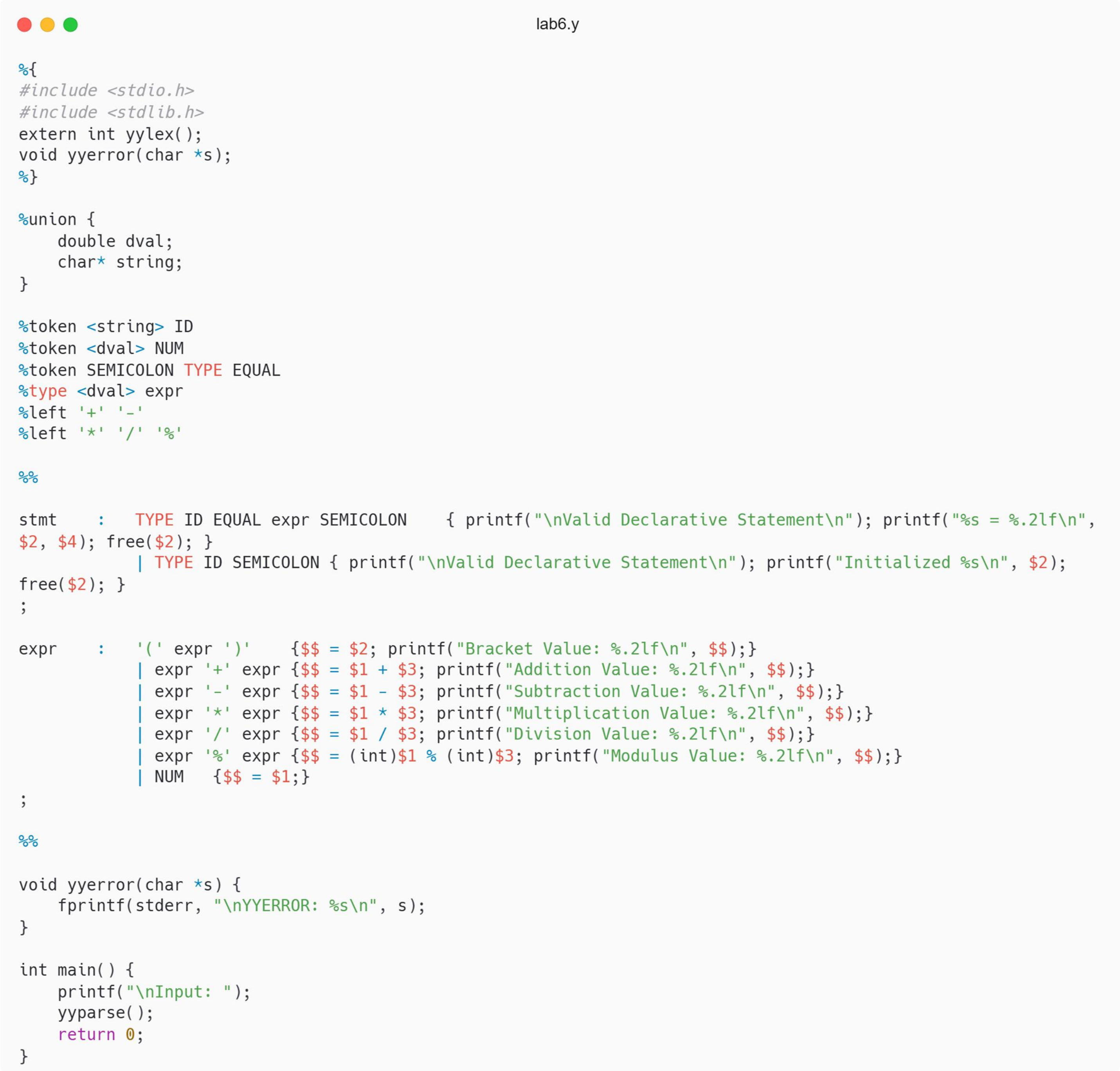


```
%{
#include <stdio.h>
#include <stdlib.h>
extern int yylex();
void yyerror(char *s);
%}
%union {
    double dval;
}
%token <dval> id num
%type <dval> expr
%left '+' '-'
%left '*' '/' '%'
%right UMINUS

%%
stmt : expr { printf("\nValid Expression\n"); };
expr : '(' expr ')' { $$ = $2; }
        | expr '+' expr { printf("\nPlus recognized"); $$ = $1 + $3; printf("\nResult: %.2lf", $$); }
        | expr '-' expr { printf("\nMinus recognized"); $$ = $1 - $3; printf("\nResult: %.2lf", $$); }
        | expr '*' expr { printf("\nMultiplication recognized"); $$ = $1 * $3; printf("\nResult: %.2lf", $$); }
        | expr '/' expr {
            printf("\nDivision recognized");
            if ($3 == 0) { printf("\nError: Division by zero!"); exit(1); }
            else { $$ = $1 / $3; printf("\nResult: %.2lf", $$); }
        }
        | expr '%' expr { printf("\nModulus recognized"); $$ = (int)$1 % (int)$3; printf("\nResult: %.2lf", $$); }
        | '-' expr %prec UMINUS { $$ = -$2; printf("\nUnary minus applied, Result: %.2lf", $$); }
        | num { $$ = $1; }
        ;
%%

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

int main() {
    printf("Enter an arithmetic expression: ");
    yyparse();
    return 0;
}
```



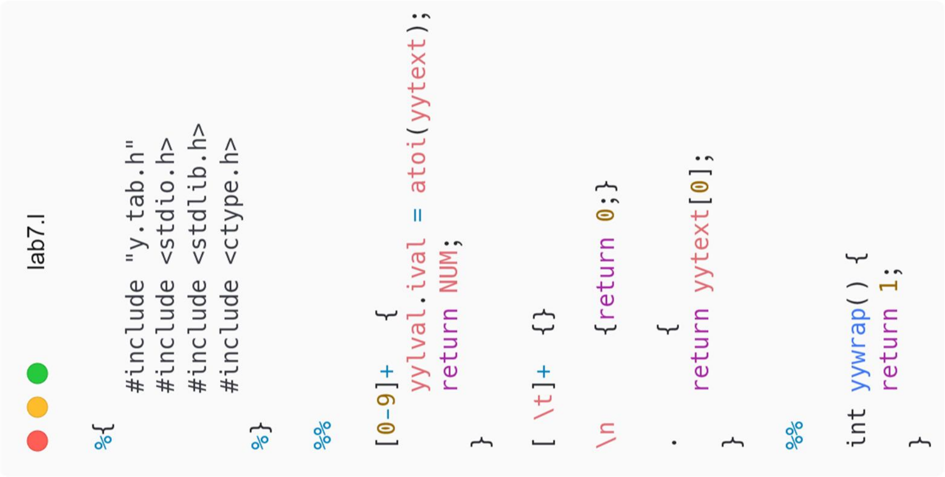
lab6.l

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

%%
"int"|"float"|"char"|"double"   {
    return TYPE;
}
[_a-zA-Z][_a-zA-Z0-9]*  {
    yylval.string = strdup(yytext);
    return ID;
}
"=" {return EQUAL;}
";" {return SEMICOLON;}
[0-9]+(\.[0-9]+)?  {
    yylval.dval = atof(yytext);
    return NUM;
}
[ \t]+  {}
\n  {return 0;}
.  {return yytext[0];}
%%

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



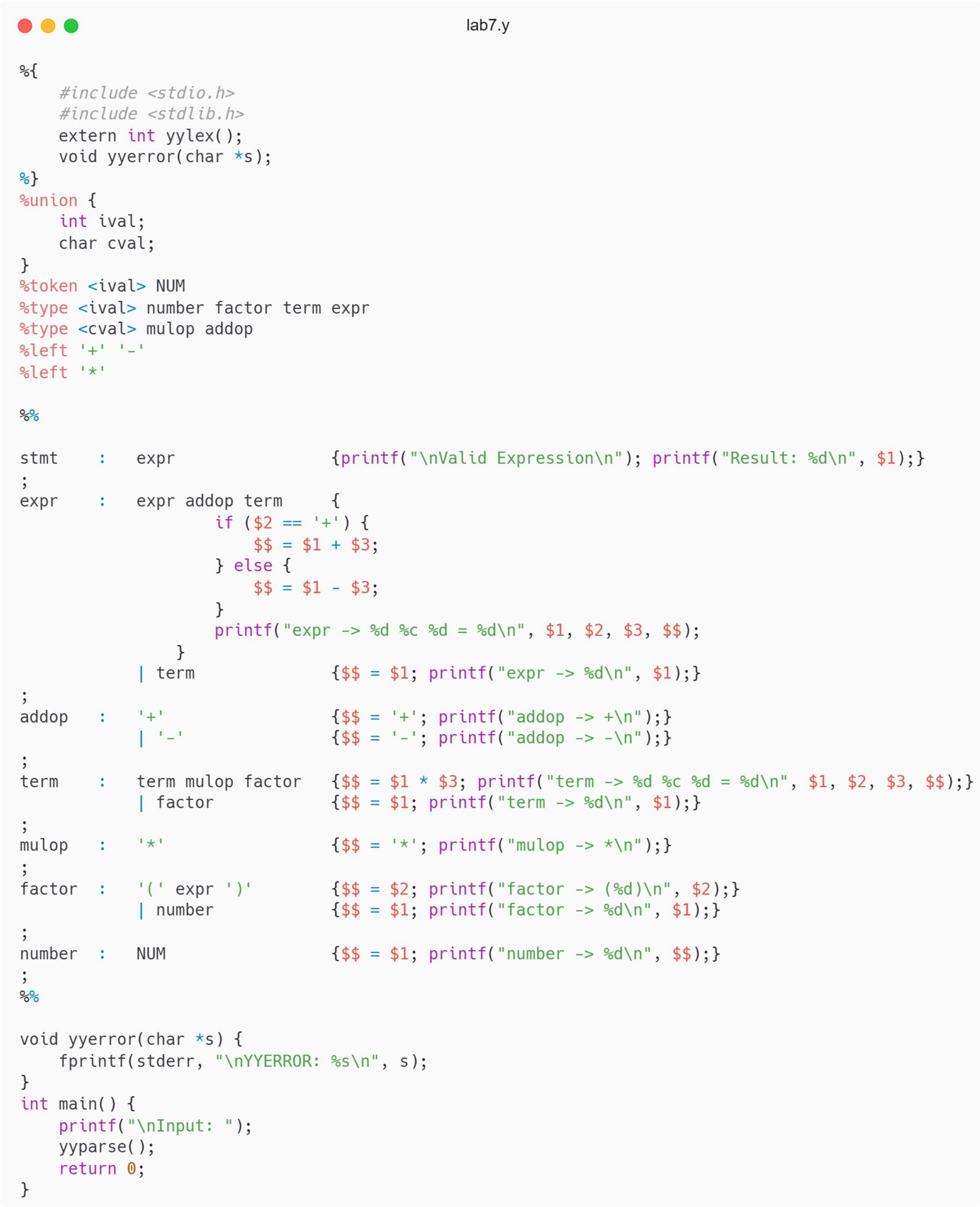
lab6.y

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

%union {
    double dval;
    char* string;
}

%token <string> ID
%token <dval> NUM
%token SEMICOLON TYPE EQUAL
%type <dval> expr
%left '+' '-'
%left '*' '/' '%'

%%

stmt    :   TYPE ID EQUAL expr SEMICOLON    { printf("\nValid Declarative Statement\n"); printf("%s = %.2lf\n",
$2, $4); free($2); }
            | TYPE ID SEMICOLON { printf("\nValid Declarative Statement\n"); printf("Initialized %s\n", $2);
free($2); }
;

expr    :   '(' expr ')'    {$$ = $2; printf("Bracket Value: %.2lf\n", $$);}
            | expr '+' expr {$$ = $1 + $3; printf("Addition Value: %.2lf\n", $$);}
            | expr '-' expr {$$ = $1 - $3; printf("Subtraction Value: %.2lf\n", $$);}
            | expr '*' expr {$$ = $1 * $3; printf("Multiplication Value: %.2lf\n", $$);}
            | expr '/' expr {$$ = $1 / $3; printf("Division Value: %.2lf\n", $$);}
            | expr '%' expr {$$ = (int)$1 % (int)$3; printf("Modulus Value: %.2lf\n", $$);}
            | NUM   {$$ = $1;}
;

%%

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

int main() {
    printf("\nInput: ");
    yyparse();
    return 0;
}
```



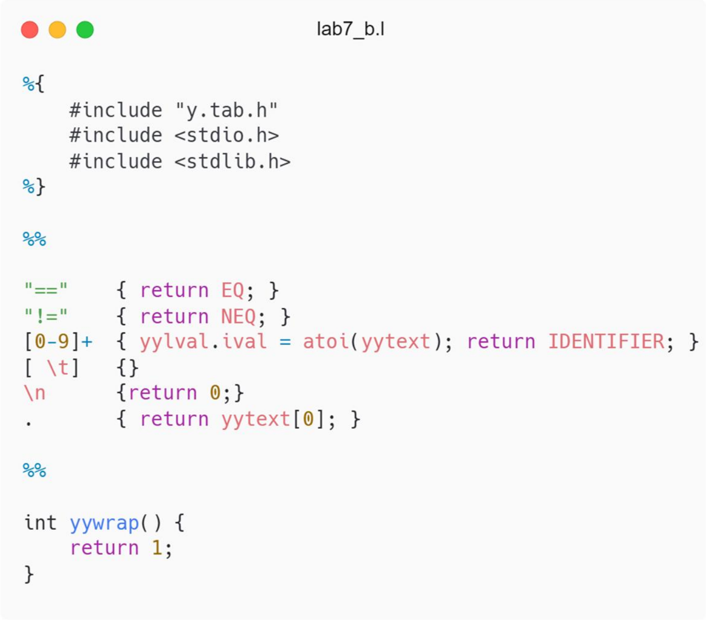
lab7.l

```
%{
    #include "y.tab.h"
    #include <stdio.h>
    #include <stdlib.h>
    #include <ctype.h>
%}
%%
[0-9]+  {
    yylval.ival = atoi(yytext);
    return NUM;
}
[ \t]+  {}
\n      {return 0;}
.       {
    return yytext[0];
}
%%
int yywrap() {
    return 1;
}
```



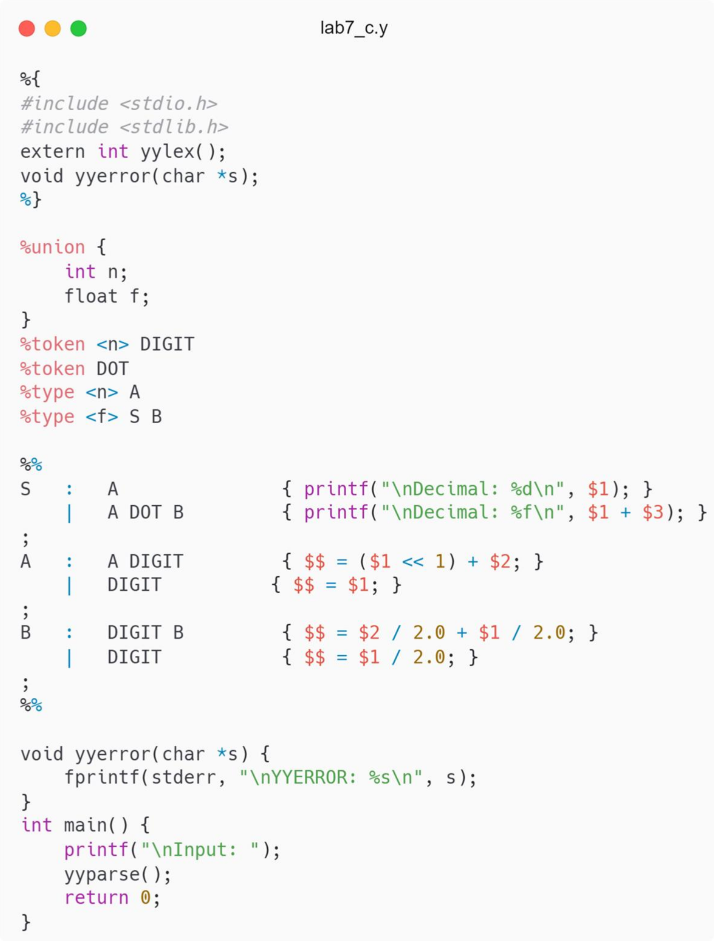
lab7.y

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    extern int yylex();
    void yyerror(char *s);
%}
%union {
    int ival;
    char cval;
}
%token <ival> NUM
%type <ival> number factor term expr
%type <cval> mulop addop
%left '+' '-'
%left '*'

%%

stmt    :   expr                {printf("\nValid Expression\n"); printf("Result: %d\n", $1);}
;
expr    :   expr addop term     {
                if ($2 == '+') {
                    $$ = $1 + $3;
                } else {
                    $$ = $1 - $3;
                }
                printf("expr -> %d %c %d = %d\n", $1, $2, $3, $$);
            }
            | term              {$$ = $1; printf("expr -> %d\n", $1);}
;
addop   :   '+'                 {$$ = '+'; printf("addop -> +\n");}
            | '-'               {$$ = '-'; printf("addop -> -\n");}
;
term    :   term mulop factor   {$$ = $1 * $3; printf("term -> %d %c %d = %d\n", $1, $2, $3, $$);}
            | factor            {$$ = $1; printf("term -> %d\n", $1);}
;
mulop   :   '*'                 {$$ = '*'; printf("mulop -> *\n");}
;
factor  :   '(' expr ')'        {$$ = $2; printf("factor -> (%d)\n", $2);}
            | number            {$$ = $1; printf("factor -> %d\n", $1);}
;
number  :   NUM                 {$$ = $1; printf("number -> %d\n", $$);}
;
%%

void yyerror(char *s) {
    fprintf(stderr, "\nYYERROR: %s\n", s);
}
int main() {
    printf("\nInput: ");
    yyparse();
    return 0;
}
```



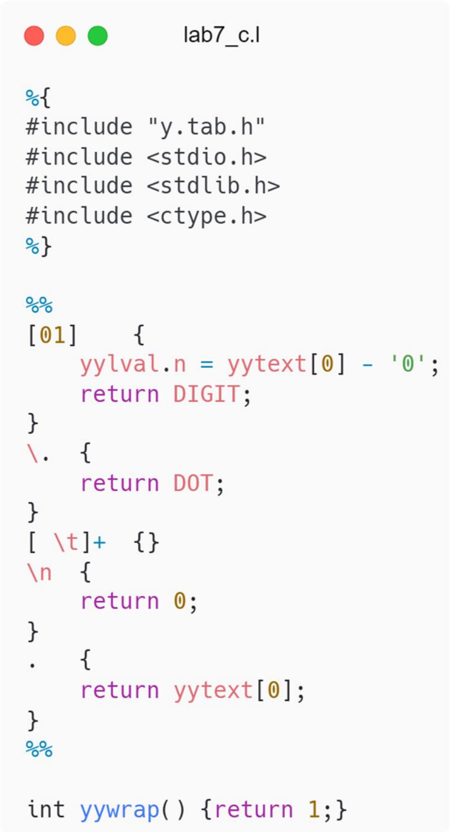
lab7_b.l

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

%%

"=="    { return EQ; }
"!="    { return NEQ; }
[0-9]+  { yylval.ival = atoi(yytext); return IDENTIFIER; }
[ \t]   {}
\n      {return 0;}
.       { return yytext[0]; }

%%

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



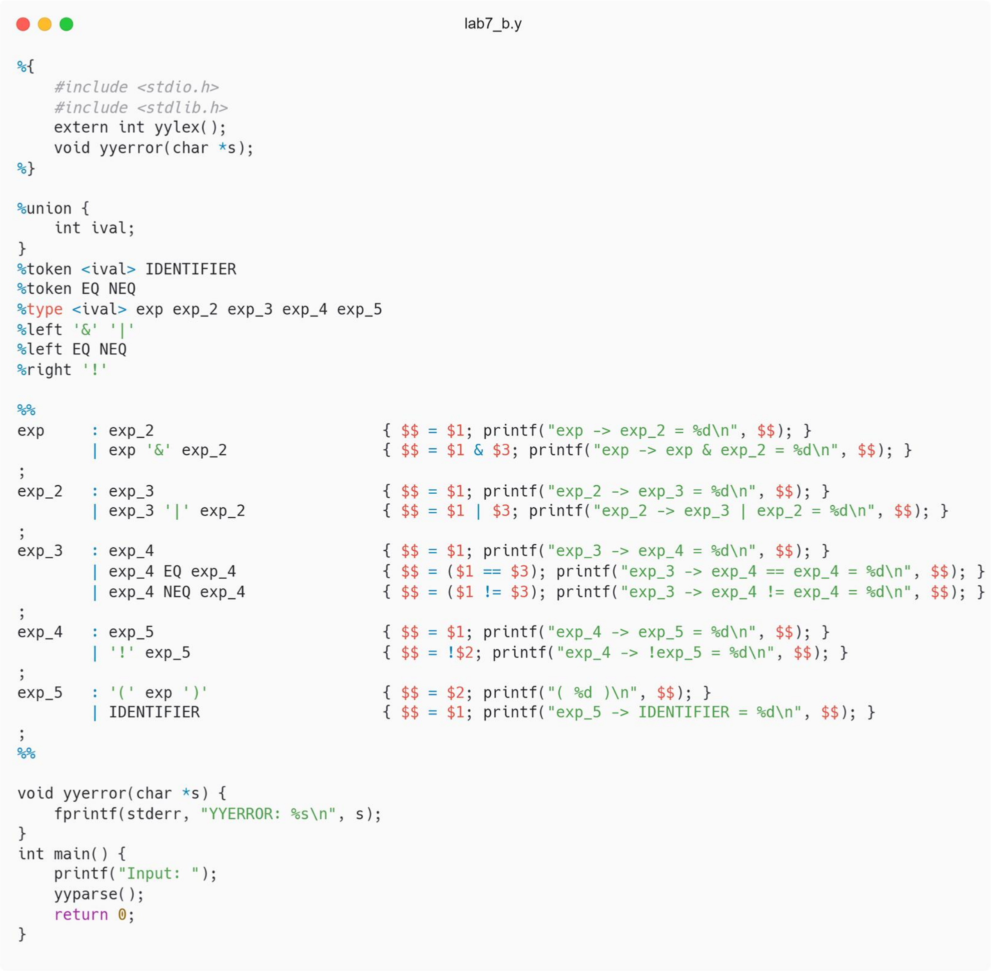
lab7_c.y

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

%union {
    int n;
    float f;
}
%token <n> DIGIT
%token DOT
%type <n> A
%type <f> S B

%%
S    :   A               { printf("\nDecimal: %d\n", $1); }
     |   A DOT B         { printf("\nDecimal: %f\n", $1 + $3); }
;
A    :   A DIGIT         { $$ = ($1 << 1) + $2; }
     |   DIGIT          { $$ = $1; }
;
B    :   DIGIT B         { $$ = $2 / 2.0 + $1 / 2.0; }
     |   DIGIT           { $$ = $1 / 2.0; }
;
%%

void yyerror(char *s) {
    fprintf(stderr, "\nYYERROR: %s\n", s);
}
int main() {
    printf("\nInput: ");
    yyparse();
    return 0;
}
```



lab7_c.l

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

%%
[01]    {
    yylval.n = yytext[0] - '0';
    return DIGIT;
}
\.  {
    return DOT;
}
[ \t]+  {}
\n  {
    return 0;
}
.   {
    return yytext[0];
}
%%

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



lab7_b.y

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

%union {
    int ival;
}
%token <ival> IDENTIFIER
%token EQ NEQ
%type <ival> exp exp_2 exp_3 exp_4 exp_5
%left '&' '|'
%left EQ NEQ
%right '!'

%%
exp     : exp_2                       { $$ = $1; printf("exp -> exp_2 = %d\n", $$); }
        | exp '&' exp_2               { $$ = $1 & $3; printf("exp -> exp & exp_2 = %d\n", $$); }
;
exp_2   : exp_3                       { $$ = $1; printf("exp_2 -> exp_3 = %d\n", $$); }
        | exp_3 '|' exp_2             { $$ = $1 | $3; printf("exp_2 -> exp_3 | exp_2 = %d\n", $$); }
;
exp_3   : exp_4                       { $$ = $1; printf("exp_3 -> exp_4 = %d\n", $$); }
        | exp_4 EQ exp_4              { $$ = ($1 == $3); printf("exp_3 -> exp_4 == exp_4 = %d\n", $$); }
        | exp_4 NEQ exp_4             { $$ = ($1 != $3); printf("exp_3 -> exp_4 != exp_4 = %d\n", $$); }
;
exp_4   : exp_5                       { $$ = $1; printf("exp_4 -> exp_5 = %d\n", $$); }
        | '!' exp_5                   { $$ = !$2; printf("exp_4 -> !exp_5 = %d\n", $$); }
;
exp_5   : '(' exp ')'                 { $$ = $2; printf("( %d )\n", $$); }
        | IDENTIFIER                  { $$ = $1; printf("exp_5 -> IDENTIFIER = %d\n", $$); }
;
%%

void yyerror(char *s) {
    fprintf(stderr, "YYERROR: %s\n", s);
}
int main() {
    printf("Input: ");
    yyparse();
    return 0;
}
```