==========Lexer.L==========

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

%%

"id"        { return ID; }
"+"         { return PLUS; }
"("         { return LPAREN; }
")"         { return RPAREN; }
"$"         { return DOLLAR; }
[ \t\n]+    ;  // Ignore whitespaces
.           { printf("Unexpected character: %s\n", yytext); }

%%

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

==========Parser.Y==========

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

    // Declare the lexical analyzer function
    int yylex(void);
    void yyerror(const char *s);
%}

%token ID PLUS LPAREN RPAREN DOLLAR

%%

S   : E DOLLAR    { printf("ACCEPT\n"); }
    ;

E   : E PLUS T    { printf("Reduce by rule: E → E + T\n"); }
    | T           { printf("Reduce by rule: E → T\n"); }
    ;

T   : LPAREN E RPAREN { printf("Reduce by rule: T → (E)\n"); }
    | ID              { printf("Reduce by rule: T → id\n"); }
    ;

%%

int main() {
    printf("Enter an expression to parse (end with $): ");
    return yyparse();  // Start the parser
}

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

OUTPUT:

```
Enter an expression to parse (end with $): id+id+id$
Reduce by rule: T → id
Reduce by rule: E → T
Reduce by rule: T → id
Reduce by rule: E → E + T
Reduce by rule: T → id
Reduce by rule: E → E + T
ACCEPT
```