A Bloody Compiler Project

Lexer (Scanner for Lexical Elements)


foo.abc
Lexer
Parser
Code Generator
foo.s

```
fn main(): int
{
    return 42;
}
```


```
foo.abc:1.1-1.3: FN 'fn' 'fn' ('fn')
foo.abc:1.4-1.8: IDENTIFIER 'main' 'main' ('IDENTIFIER')
foo.abc:1.8-1.9: LPAREN '(' '(' ('(')
foo.abc:1.9-1.10: RPAREN ')' ')' (')')
foo.abc:1.10-1.11: COLON ':' ':' (':')
foo.abc:1.12-1.15: IDENTIFIER 'int' 'int' ('IDENTIFIER')
foo.abc:2.1-2.2: LBRACE '{' '{' ('{')
foo.abc:3.5-3.11: RETURN 'return' 'return' ('return')
foo.abc:3.12-3.14: DECIMAL_LITERAL '42' '42' ('DECIMAL_LITERAL')
foo.abc:3.14-3.15: SEMICOLON ';' ';' (';')
foo.abc:4.1-4.2: RBRACE '}' '}' ('}')
foo.abc:5.1-5.1: EOI '' '' ('EOI')
```

```
fn main(): int
{
    return 42;
}
```


```
foo.abc:1.1-1.3: FN 'fn' 'fn' ('fn')
foo.abc:1.4-1.8: IDENTIFIER 'main' 'main' ('IDENTIFIER')
foo.abc:1.8-1.9: LPAREN '(' '(' ('(')
foo.abc:1.9-1.10: RPAREN ')' ')' (')')
foo.abc:1.10-1.11: COLON ':' ':' (':')
foo.abc:1.12-1.15: IDENTIFIER 'int' 'int' ('IDENTIFIER')
foo.abc:2.1-2.2: LBRACE '{' '{' ('{')
foo.abc:3.5-3.11: RETURN 'return' 'return' ('return')
foo.abc:3.12-3.14: DECIMAL_LITERAL '42' '42' ('DECIMAL_LITERAL')
foo.abc:3.14-3.15: SEMICOLON ';' ';' (';')
foo.abc:4.1-4.2: RBRACE '}' '}' ('}')
foo.abc:5.1-5.1: EOI '' '' ('EOI')
```

```
12 + 432 + +
+ 34 ()
```


```
DECIMAL_LITERAL
PLUS
DECIMAL_LITERAL
PLUS
PLUS
PLUS
DECIMAL_LITERAL
BAD_TOKEN
BAD_TOKEN
```

```
@ <stdio.hdr>

enum TokenKind
{
    BAD_TOKEN,
    EOI,
    DECIMAL_LITERAL,
    PLUS,
};

global token: TokenKind = BAD_TOKEN;

fn getToken(): TokenKind;

// -- lexer implementation

/* TODO */

// -- simple lexer test

fn main()
{
    /* TODO */
}
```

```
@ <stdio.hdr>

enum TokenKind
{
    BAD_TOKEN,
    EOI,
    DECIMAL_LITERAL,
    PLUS,
};

global token: TokenKind = BAD_TOKEN;

fn getToken(): TokenKind;

//-- lexer implementation

/* TODO */

//-- simple lexer test

fn main()
{
    while (getToken() != EOI) {
        if (token == DECIMAL_LITERAL) {
            printf("DECIMAL_LITERAL\n");
        } else if (token == PLUS) {
            printf("PLUS\n");
        } else if (token == BAD_TOKEN) {
            printf("BAD_TOKEN\n");
        } else {
            printf("token not handled! token = %d\n", token);
        }
    }
}
```

```
@ <stdio.hdr>

enum TokenKind
{
    BAD_TOKEN,
    EOI,
    DECIMAL_LITERAL,
    PLUS,
};

global token: TokenKind = BAD_TOKEN;

fn getToken(): TokenKind;

//-- lexer implementation

fn getToken(): TokenKind
{
    if (token == BAD_TOKEN) {
        return token = PLUS;
    }
    return token = EOI;
}

//-- simple lexer test

fn main()
{
    while (getToken() != EOI) {
        if (token == DECIMAL_LITERAL) {
            printf("DECIMAL_LITERAL\n");
        } else if (token == PLUS) {
            printf("PLUS\n");
        } else if (token == BAD_TOKEN) {
            printf("BAD_TOKEN\n");
        } else {
            printf("token not handled! token = %d\n", token);
        }
    }
}
```

```
@ <stdio.hdr>

enum TokenKind
{
    BAD_TOKEN,
    EOI,
    DECIMAL_LITERAL,
    PLUS,
};

global token: TokenKind = BAD_TOKEN;

fn getToken(): TokenKind;

//-- lexer implementation

/* TODO */

//-- simple lexer test

fn main()
{
    while (getToken() != EOI) {
        if (token == DECIMAL_LITERAL) {
            printf("DECIMAL_LITERAL\n");
        } else if (token == PLUS) {
            printf("PLUS\n");
        } else if (token == BAD_TOKEN) {
            printf("BAD_TOKEN\n");
        } else {
            printf("token not handled! token = %d\n", token);
        }
    }
}
```

```
global ch: int;

fn getToken(): TokenKind
{
    while (ch == 0 || ch == ' ' || ch == '\t' || ch == '\n') {
        ch = getchar();
    }
    if (ch >= '0' && ch <= '9') {
        while (ch >= '0' && ch <= '9') {
            ch = getchar();
        }
        return token = DECIMAL_LITERAL;
    } else if (ch == '+') {
        ch = getchar();
        return token = PLUS;
    } else if (ch == EOF) {
        return token = EOI;
    } else {
        ch = getchar();
        return token = BAD_TOKEN;
    }
}
```

```
global ch: int;

fn getToken(): TokenKind
{
    while (ch == 0 || ch == ' ' || ch == '\t' || ch == '\n') {
        ch = getchar();
    }
    if (ch >= '0' && ch <= '9') {
        while (ch >= '0' && ch <= '9') {
            ch = getchar();
        }
        return token = DECIMAL_LITERAL;
    } else if (ch == '+') {
        ch = getchar();
        return token = PLUS;
    } else if (ch == EOF) {
        return token = EOI;
    } else {
        ch = getchar();
        return token = BAD_TOKEN;
    }
}
```