A Bloody Compiler Project

Parser and Code Generator

EBNF (Extended Backus-Nauer Form)

```
decimal-literal = digit { digit }

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

EBNF (Extended Backus-Nauer Form)

```
expr = term { "+" term }

term = decimal-literal
```

```
fn parseExpr(): bool
{
    // ...
}

fn parseTerm(): bool
{
    // ...
}
```

```
expr = term { "+" term }

term = decimal-literal
```

```
fn parseTerm(): bool
{
    if (token.kind == DECIMAL_LITERAL) {
        getToken();
        return true;
    } else {
        return false;
    }
}
```

```
expr = term { "+" term }


term = decimal-literal
```

```
fn parseExpr(): bool
{
    if (!parseTerm()) {
        return false;
    }
    while (token.kind == PLUS) {
        getToken();
        if (!parseTerm()) {
            syntaxError();
        }
    }
    return true;
}
```

```
expr = term { "+" term }

term = decimal-literal
```

```
fn main()
{
    getToken();
    if (parseExpr() && token.kind == EOI) {
        printf("ok!\n");
    } else {
        syntaxError();
    }
}
```

```
expr = term { "+" term }

term = decimal-literal
```

```
@ <stdio.hdr>
@ <stdlib.hdr>
@ "lexer.hdr"

fn syntaxError()
{
    printf("syntax error\n");
    exit(1);
}

fn parseTerm(): bool;

fn parseExpr(): bool
{
    // ...
}

fn parseTerm(): bool
{
    // ...
}

fn main()
{
    // ...
}
```

Demo: Parsing some valid and invalid expressions

```
expr = term { "+" term }


term = decimal-literal
```

```
fn getReg(): int
{
    static reg: int = 1;
    return reg++;
}

// ...
```

```
expr = term { "+" term }

term = decimal-literal
```

```
fn getReg(): int
{
    static reg: int = 1;
    return reg++;
}

// ...

fn parseTerm(): int
{
    if (token.kind == DECIMAL_LITERAL) {
        local reg: int = getReg();
        printf("\tload %s, %%%d\n", token.val, reg);
        getToken();
        return reg;
    } else {
        return -1;
    }
}
```

```
expr = term { "+" term }

term = decimal-literal
```

```
fn parseExpr(): int
{
    local left: int = parseTerm();
    if (left < 0) {
        return -1;
    }
    while (token.kind == PLUS) {
        getToken();
        local right: int = parseTerm();
        if (right < 0) {
            syntaxError();
        }
        local dest: int = getReg();
        printf("\tadd %%%d, %%%d, %%%d\n", left, right, dest);
        left = dest;
    }
    return left;
}
```

```
expr = term { "+" term }


term = decimal-literal
```

```
fn main()
{
    getToken();
    local reg: int = parseExpr();
    if (reg >= 0 && token.kind == EOI) {
        printf("\thalt %%%d\n", reg);
    } else {
        syntaxError();
    }
}
```

```
expr = term { "+" term }

term = decimal-literal
```

```
@ <stdio.hdr>
@ <stdlib.hdr>
@ "lexer.hdr"

fn syntaxError()
{
    // ...
}

fn getReg(): int
{
    // ...
}

fn parseTerm(): int;

fn parseExpr(): int
{
    // ...
}

fn parseTerm(): int
{
    // ...
}

fn main()
{
    // ...
}
```

Demo: Generating some code …

```
expr = term { "+" term }

term = factor { "*" factor }

factor = decimal-literal
       | "(" expr ")"
```

```
expr = term { "+" term }

term = factor { "*" factor }

factor = decimal-literal
       | "(" expr ")"
```

```
fn parseExpr(): int
{
    local left: int = parseTerm();
    if (left < 0) {
        return -1;
    }
    while (token.kind == PLUS) {
        getToken();
        local right: int = parseTerm();
        if (right < 0) {
            syntaxError();
        }
        local dest: int = getReg();
        printf("\tadd %%%d, %%%d, %%%d\n", left, right, dest);
        left = dest;
    }
    return left;
}
```

```
expr = term { "+" term }

term = factor { "*" factor }

factor = decimal-literal
       | "(" expr ")"
```

```
fn parseTerm(): int
{
    local left: int = parseFactor();
    if (left < 0) {
        return -1;
    }
    while (token.kind == ASTERISK) {
        getToken();
        local right: int = parseFactor();
        if (right < 0) {
            syntaxError();
        }
        local dest: int = getReg();
        printf("\tmul %%%d, %%%d, %%%d\n", left, right, dest);
        left = dest;
    }
    return left;
}
```

```
expr = term { "+" term }

term = factor { "*" factor }

factor = decimal-literal
       | "(" expr ")"
```

```
fn parseFactor(): int
{
    if (token.kind == DECIMAL_LITERAL) {
        local reg: int = getReg();
        printf("\tload %s, %%%d\n", token.val, reg);
        getToken();
        return reg;
    } else if (token.kind == LPAREN) {
        getToken();
        local reg: int = parseExpr();
        if (reg < 0) {
            syntaxError();
        }
        if (token.kind != RPAREN) {
            syntaxError();
        }
        getToken();
        return reg;
    } else {
        return -1;
    }
}
```