Compilers Lab 4 Report

106119029 Dipesh Kafle, 106119073 Marmik Upadhyay, 106119117 Sobhagya Singh Dewal

Contents

Project Link

Project Link

Description

- We are using Recursive Descent Parser for the project.
- The expression parsing is done through Pratt Parser(A Recursive Operator Precedence Parser).

Operator Precedence

- lhs > rhs means right associative.
- rhs > lhs means left associative.

```
19  static const inline std::unordered_map<TokenType, int> prefix_binding_power{ ▸ 2 refs
18      {TokenType::LEFT_PAREN, 0},
17      {TokenType::BANG, 15},
16      {TokenType::PLUS, 16},
15      {TokenType::MINUS, 16},
14
13  };
12
11  // We model associativity with diff left and right binding power values
10  // If lbp > rbp, means the operator is right associative, else left
 9  // associative
 8  static const inline std::unordered_map<TokenType, std::pair<int, int>>
 7      infix_binding_power{ ▸ 2 refs
 6          {TokenType::EQUAL, {2, 1}},
 5          {TokenType::IF, {4, 3}},
 4          {TokenType::OR, {5, 6}},
 3          {TokenType::AND, {7, 8}},
 2          {TokenType::EQUAL_EQUAL, {9, 10}},
 1          {TokenType::BANG_EQUAL, {9, 10}},
56          {TokenType::LESS, {11, 12}},
 1          {TokenType::LESS_EQUAL, {11, 12}},
 2          {TokenType::GREATER, {11, 12}},
 3          {TokenType::GREATER_EQUAL, {11, 12}},
 4          {TokenType::PLUS, {13, 14}},
 5          {TokenType::MINUS, {13, 14}},
 6          {TokenType::SLASH, {15, 16}},
 7          {TokenType::STAR, {15, 16}},
 8          {TokenType::DOT, {19, 20}},
 9          {TokenType::LEFT_PAREN, {21, 22}},
10      };
11
```

Parser Class

- Recursive Descent based parser

```
  static parse_error error(const Token &tok, const char *err_msg); ▸ 0 ref 0 ref 7 refs

  expr_or_err finish_call(std::unique_ptr<expr::Expr> callee); ▸ 1 ref 0 ref

  expr_or_err expression(int binding_power = 0); ▸ 0 ref 12 refs

  expr_or_err prefix_expression(); ▸ 1 ref

  stmt_or_err statement(); ▸ 4 refs
  stmt_or_err break_statement(); ▸ 1 ref
  stmt_or_err continue_statement(); ▸ 1 ref
  stmt_or_err expression_statement(); ▸ 1 ref
  stmt_or_err print_statement(bool new_line); ▸ 2 refs 0 ref
  stmt_or_err return_statement(); ▸ 1 ref
  stmt_or_err declaration(); ▸ 2 refs
  stmt_or_err fn_declaration(const std::string &type); ▸ 1 ref 0 ref
  stmt_or_err let_declaration(); ▸ 1 ref
  stmt_or_err if_statement(); ▸ 1 ref
  stmt_or_err while_statement(); ▸ 1 ref
  // stmt_or_err for_statement();
  stmt_or_err data_definition(); ▸ 1 ref

  tl::expected<std::vector<std::unique_ptr<Stmt>>, parse_error> block(); ▸ 2 refs

public:
  explicit Parser(std::vector<Token> _toks); ▸ 3 refs 0 ref
  expr_or_err parse_expr(); ▸ 2 refs
  std::vector<stmt_or_err> parse(); ▸ 1 ref
};
```

Output

Error Handling

```
Enma/build on  main [$?] via △ v3.22.3
λ cat ../examples/continue_error.enma

continue;

Enma/build on  main [$?] via △ v3.22.3
λ ./bin/enma-frontend ../examples/continue_error.enma
[line 2] Error  at ';' : Continue statement is not inside any form of loop. This is not valid
Enma/build on  main [$?] via △ v3.22.3
λ cat ../examples/break_error.enma

break;
Enma/build on  main [$?] via △ v3.22.3
λ ./bin/enma-frontend ../examples/break_error.enma
[line 2] Error  at ';' : Break statement is not inside any form of loop. This is not valid
Enma/build on  main [$?] via △ v3.22.3
λ cat ../examples/continue_and_break.enma

let i: int =0;

while(i<10){
        if (i<5){
                continue;
        }
        break;
}
Enma/build on  main [$?] via △ v3.22.3
λ ./bin/enma-frontend ../examples/continue_and_break.enma
(Let (i : int) 0)

(While cond (< i 10) do (Block [ (If (< i 5) then (Block [ (continue) ]) else {})); (break) ]))

Enma/build on  main [$?] via △ v3.22.3
λ
```

```
Enma/build on  main [$x?] via  v3.22.3
λ cat ../examples/type_miss_error.enma
let x = 10;
let y : = 20;
Enma/build on  main [$x?] via  v3.22.3
λ ./bin/enma-frontend ../examples/type_miss_error.enma
[line 1] Error  at '=' : Expected colon after identifier
[line 2] Error  at '=' : Expected Type in let declartion
Enma/build on  main [$x?] via  v3.22.3
λ
```

Parsing a full program

- Program

```
Enma/build on  main [$x?] via  v3.22.3
λ cat ../examples/input.enma
let z : string = "abc";

data Position = {
        x: int;
        y: int;
}

let x : int = 10;

fn is_origin(pos: Position ){
        if (pos.x == 0 and pos.y == 0 ){
                true;
        }else{
                !true;
        }
}

// This is error
//$$

fn sum(n: int ){
        let i : int = 0;
        let sm : int = 0;
        while( i ⩽ n){
                sm = sm + i;
                i =  i + 1;
        }
        sm;
}
```

- After Parsing

```
Enma/build on  main [$x?] via  v3.22.3
λ ./bin/enma-frontend ../examples/input.enma
(Let (z : string) abc)

(DataDefn Position [x: int,y: int])

(Let (x : int) 10)

(Fn  def is_origin [pos: Position] [ (If (and (== (. pos x) 0) (== (. pos y) 0)) then (Block [ true ]) else (Block [ (! true) ])) ] )

(Fn  def sum [n: int] [ (Let (i : int) 0); (Let (sm : int) 0); (While cond (⩽ i n) do (Block [ (= sm (+ sm i)); (= i (+ i 1)) ])); sm ] )

Enma/build on  main [$x?] via  v3.22.3
λ
```