Swimft

Alunos: Beatriz e Vitor

Data: 28/06/2019

Universidade Federal Fluminense

Objetivos desta apresentação

- Apresentação de conclusão da segunda etapa do trabalho

O que foi feito

- Lexer adaptado para Imp-1
- Parser para Imp-1
- Pi Framework: implementado compilador com ambientes e declarações (variáveis e constantes)
- Print()
- Refatoração do código: limpeza visual

O que foi feito: Lexer

```
case INITIALIZER                ("=", { _ in ImpToken.INITIALIZER }),
case REFERENCE(String)          ("(&|\\(\\*)", { (m: String) in ImpToken.REFERENCE(m) }),
case LET
case IN
case DECLARATION(String)        ("(?![0-9])[a-zA-Z_][a-zA-Z_0-9]*", { (m: String) in matchName(string: m) }),
case PRINT
```

Figura 1: enum que define os tokens para Imp-1 - imp token processor

O que foi feito: Lexer

```
else if string == "let"
{
        return ImpToken.LET
}
else if string == "var" || string == "cons"
{
        return ImpToken.DECLARATION(string)
}
else if string == "in"
{
        return ImpToken.IN
}
else if string == "print"
{
        return ImpToken.PRINT
}
else
{
        return ImpToken.IDENTIFIER(string)
}
```

Figura 2: função matchName

O que foi feito: Parser

```
/// - This wrap the reference operations(<reference>).
public protocol ReferenceImpNode: ExpressionImpNode
{
        var identifier: IdentifierImpNode { get }
}
/// - This wrap the address reference operation(<address_reference>).
public struct AddressReferenceImpNode: ReferenceImpNode
{
        public let identifier: IdentifierImpNode
        public var description: String
        {
                return "AddressReferenceNode(\(identifier))"
        }
}

/// - This wrap the value reference operation(<value_reference>).
public struct ValueReferenceImpNode: ReferenceImpNode, LogicalExpressionImpNode, ArithmeticExpressionImpNode
{
        public let identifier: IdentifierImpNode
        public var description: String
        {
                return "ValueReferenceNode(\(identifier))"
        }
}
```

Figura 3: estrutura do nó imp - Reference

O que foi feito: Parser

```
/// - Helper function for dealing with the REFERENCE imp token processing(<reference>).
/// - Return
///     - The relative reference imp node to the given token.
private func parseReference () throws -> ExpressionImpNode
{
        guard case let ImpToken.REFERENCE(op) = tokens.pop() else
        {
                throw ParserError.ExpectedToken("ImpToken.REFERENCE")
        }

        let identifier: IdentifierImpNode = try parseIdentifier()

        switch(op)
        {
                case "&":
                        return AddressReferenceImpNode(identifier: identifier)
                case "(*":
                        guard case ImpToken.BRACKET_RIGHT = tokens.pop() else
                        {
                                throw ParserError.ExpectedToken("ImpToken.BRACKET_RIGHT")
                        }
                        return ValueReferenceImpNode(identifier: identifier)
                default:
                        throw ImpParserError.UndefinedOperator(op)
        }
}
```

Figura 4: processamento de Reference token com retorno do nó imp específico

O que foi feito: Parser

```
/// - This wrap the variable node(<variable_declaration>).
public struct VariableDeclarationImpNode: DeclarationImpNode
{
        let identifier: IdentifierImpNode
        let expression: ExpressionImpNode

        public var description: String
        {
                return "VariableNode(\(identifier), \(expression))"
        }
}

/// - This wrap the constant node(<constant_declaration>).
public struct ConstantDeclarationImpNode: DeclarationImpNode
{
        let identifier: IdentifierImpNode
        let expression: ExpressionImpNode

        public var description: String
        {
                return "ConstantNode(\(identifier), \(expression))"
        }
}
```

Figura 5: estrutura do nó imp - Declaration

O que foi feito: Parser

```
/// - Helper function for dealing with the DECLARATION imp token processing(<declaration>).
///     Also here all its ramifications will be processed(<variable_declaration>, <constant_declaration>))
/// - Return
///     - The relative declaration imp node to the given token.
private func parseDeclaration () throws -> DeclarationImpNode
{
        guard case let ImpToken.DECLARATION(op) = tokens.pop() else
        {
                throw ParserError.ExpectedToken("ImpToken.DECLARATION")
        }
        let identifier: IdentifierImpNode = try parseIdentifier()

        guard case ImpToken.INITIALIZER = tokens.pop() else
        {
                throw ParserError.ExpectedToken("ImpToken.INITIALIZER")
        }

        let expression: ExpressionImpNode = try parseExpression()
        switch(op)
        {
                case "var":
                        return VariableDeclarationImpNode(identifier: identifier, expression: expression)
                case "cons":
                        return ConstantDeclarationImpNode(identifier: identifier, expression: expression)
                default:
                        throw ImpParserError.UndefinedOperator(op)
        }
}
```

Figura 6: processamento de Declaration token com retorno do nó imp específico

O que foi feito: Parser

```
public struct BlockImpNode: CommandImpNode
{
        let declaration: [DeclarationImpNode]
        let command: [CommandImpNode]
        public var description: String
        {
                return "BlockNode([\(declaration) - \(declaration.count)], [\(command) - \(command.count)])"
        }
}

/// - This wrap the print operation(<print>).
public struct PrintImpNode: CommandImpNode
{
        let expression: ExpressionImpNode
        public var description: String
        {
                return "PrintNode(\(expression))"
        }
}
```

Figura 7: estrutura do nó imp - Block e Print

O que foi feito: Parser

```
/// - Helper function for dealing with the PRINT imp token processing(<print>).
/// - Return
///     - The relative print imp node to the given token.
private func parsePrint () throws -> PrintImpNode
{
        guard case ImpToken.PRINT = tokens.pop() else
        {
                throw ParserError.ExpectedToken("ImpToken.PRINT")
        }

        guard case ImpToken.BRACKET_LEFT = tokens.pop() else
        {
                throw ParserError.ExpectedToken("ImpToken.BRACKET_LEFT")
        }

        let expression: ExpressionImpNode = try parseExpression()

        guard case ImpToken.BRACKET_RIGHT = tokens.pop() else
        {
                throw ParserError.ExpectedToken("ImpToken.BRACKET_RIGHT")
        }

        return PrintImpNode(expression: expression)
}
```

Figura 8: processamento de Print token

O que foi feito: Parser

```
/// - Helper function for dealing with the BLOCK imp token processing(<block>).
/// - Return
///     - The relative block imp node to the given token.
private func parseBlock () throws -> BlockImpNode
{
        guard case ImpToken.LET = tokens.pop() else
        {
                throw ParserError.ExpectedToken("ImpToken.LET")
        }

        var declarationForest: [DeclarationImpNode] = [DeclarationImpNode]()
        while(true)
        {
                let declaration: DeclarationImpNode = try parseDeclaration()
                declarationForest.append(declaration)
                if (tokens.isEmpty())
                {
                        throw ParserError.ExpectedToken("ImpToken.IN")
                }
                else if case ImpToken.IN = tokens.peek()
                {
                        tokens.skip()
                        break
                }
                else if case ImpToken.COMMA = tokens.peek()
                {
                        tokens.skip()
                }
        }
```

Figura 9: processamento de Block token

O que foi feito: Parser

```
        var commandForest: [CommandImpNode] = [CommandImpNode]()
        while true
        {
                let command: CommandImpNode = try parseGrammar()
                commandForest.append(command)
                if tokens.isEmpty()
                {
                        throw ParserError.ExpectedToken("ImpToken.END")
                }
                else if case ImpToken.END = tokens.peek()
                {
                        tokens.skip()
                        break
                }
        }
        return BlockImpNode(declaration: declarationForest, command: commandForest)
}
```

Figura 10: continuação... processamento de Block token

O que foi feito: Pi Framework

- Vamos ao código...

O que não foi feito e porquê

- Código completamente documentado

Avaliação da evolução do trabalho

- Here we go to Imp-2!