

UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

ARTHUR PONTES DE MIRANDA SOARES
BEATRIZ RODRIGUES CAVALCANTE
GABRIEL GOMES DE OLIVEIRA
DAVI DA SILVA ROMÃO

**PROJETO DE INTERPRETADOR PARA A LINGUAGEM MINIPAR
COMPILADORES**

Relatório apresentado como requisito parcial
para aprovação na disciplina Compiladores,
ministrada pelo Prof. Dr. Arturo Hernández
Domínguez, do curso de Ciência da
Computação, na Universidade Federal de
Alagoas.

Sumário

| | |
|----------------------------------------------------|----|
| Sumário..... | 2 |
| 1 INTRODUÇÃO..... | 3 |
| 2 REQUISITOS..... | 4 |
| 3 TECNOLOGIAS..... | 6 |
| 4 GRAMÁTICA..... | 7 |
| 5 ARQUITETURA..... | 10 |
| 6 PSEUDOCÓDIGOS..... | 12 |
| 6.1 Analisador Léxico..... | 12 |
| 6.2 Analisador Sintático..... | 13 |
| 6.3 Tabela de Símbolos e Variáveis..... | 15 |
| 6.4 Analisador Semântico..... | 16 |
| 6.5 Executor de código..... | 20 |
| 7 TESTES..... | 24 |
| 7.1 Neurônio..... | 24 |
| 7.2 Quicksort..... | 26 |
| 7.3 Rede Neural..... | 27 |
| 7.4 Recomendação..... | 29 |
| 7.5 Cliente - Servidor..... | 32 |
| 7.5.1 Servidor em Java com Minipar Cliente..... | 33 |
| 7.6 Execução paralela do Fatorial e Fibonacci..... | 35 |
| 7.7 Fatorial recursivo..... | 36 |
| 7.8 Recomendação 2..... | 37 |
| 8 INTERFACE..... | 41 |
| 9 HIGHLIGHT SYNTAX..... | 44 |
| 10 LINK..... | 45 |

1 INTRODUÇÃO

Minipar é uma linguagem de programação que possibilita a execução de blocos de código de forma sequencial ou paralela de forma nativa por meio de funções internas `seq()` e `par()`. Além disso, permite a execução concorrente de blocos de código e a criação simplificada de conexões por meio de sockets para compartilhamento da carga de trabalho de programas.

A linguagem foi criada para auxiliar estudantes na compreensão de conceitos ligados à construção de compiladores e interpretadores, tendo sido inicialmente desenvolvida como parte do processo avaliativo de disciplinas de Compiladores. A cada novo período, os alunos se deparam com a tarefa de adicionar novas funcionalidades e expandir as capacidades do interpretador do Minipar.

Este relatório apresenta os detalhes de implementação do interpretador da linguagem Minipar que incluem as novas funcionalidades especificadas no período corrente. Essas funcionalidades incluem estruturas de dados nativas, como listas e dicionários, *list comprehensions*, para simplificar a construção de novas listas a partir de outras, acesso a matrizes (listas aninhadas) e operação de *slicing*. Com essas novas funcionalidades, programas mais complexos como uma rede neural simples e um sistema de recomendações puderam ser criados na linguagem Minipar. O interpretador foi desenvolvido com base em componentes de software e o planejamento e execução do projeto foi feito com base na metodologia ágil SCRUM.

2 REQUISITOS

Como parte da organização e execução do projeto de desenvolvimento do interpretador para a linguagem Minipar, adotou-se o framework SCRUM, visando uma abordagem iterativa e incremental. O projeto foi dividido em funcionalidades organizadas na Tabela 1, que descreve todas as histórias de usuário e tarefas necessárias para a entrega do sistema completo, incluindo análise léxica, sintática, semântica, execução paralela e interface web. A partir desse conjunto de funcionalidades, foi planejada a Sprint de desenvolvimento. Os detalhes das tarefas selecionadas para essa sprint estão descritos na Tabela 2, contendo as estimativas de esforço, responsáveis e status atual de cada item.

| ID | História do Usuário | Prioridade | Observações |
|----|---------------------------------------------------------------------------------------|------------|------------------------------------------------|
| 1 | Definir a gramática da linguagem Minipar para servir de base ao analisador sintático. | Alta | Base para o parser |
| 2 | Implementar o analisador léxico para identificar corretamente os tokens da linguagem. | Alta | Deve ser testado com vários casos |
| 3 | Implementar o analisador sintático baseado na gramática definida para gerar a AST. | Alta | Depende da tarefa 1 |
| 4 | Implementar o analisador semântico para verificar tipos e estruturas da linguagem. | Alta | Usa a AST como entrada |
| 5 | Digitar um código Minipar e ver o resultado de sua execução. | Alta | Depende de toda a cadeia de análise e execução |
| 6 | Implementar o módulo de execução com suporte a blocos paralelos. | Alta | Parte mais complexa |
| 7 | Criar uma interface web simples com FastAPI que aceite código e mostre o resultado. | Média | Pode ser feita com templates simples |
| 8 | Implementar mensagens de erro amigáveis quando o código estiver incorreto. | Média | Cobre erros léxicos, sintáticos e semânticos |
| 9 | Integrar todos os módulos. | Alta | Usar uma única classe |

Tabela 1: Product Backlog

| ID | Tarefa | Início | Fim |
|-----------|--------------------------------------------------------------|---------------|------------|
| 1 | Preparar repositório com estrutura inicial do projeto | 22/04/2025 | 22/04/2025 |
| 2 | Definir a gramática formal da linguagem Minipar | 23/04/2025 | 23/04/2025 |
| 3 | Implementar o analisador léxico com reconhecimento de tokens | 24/04/2025 | 25/04/2025 |
| 4 | Especificar estrutura de dados da AST | 26/04/2025 | 26/04/2025 |
| 5 | Implementar o Parser | 27/04/2025 | 30/05/2025 |
| 6 | Implementar o Analisador Semântico | 01/05/2025 | 02/05/2025 |
| 7 | Implementar o Executador de Código | 01/05/2025 | 03/05/2025 |
| 8 | Integração dos módulos | 04/05/2025 | 04/05/2025 |
| 9 | Implementação da Interface Web | 03/05/2025 | 04/05/2025 |
| 10 | Teste exploratórios | 05/05/2025 | 06/05/2025 |
| 11 | Escrever relatório | 05/05/2025 | 06/05/2025 |

Tabela 2: Sprint Backlog

3 TECNOLOGIAS

Para o desenvolvimento do interpretador foi usado Python, sem bibliotecas externas auxiliares. Quanto ao desenvolvimento da interface, foi utilizado o FastAPI em conjunto com um sistema de templates, construídos com HTML, CSS e JavaScript.

A escolha da linguagem foi motivada pela facilidade de desenvolvimento de projetos utilizando orientação a objetos, além do suporte de bibliotecas internas da linguagem para lidar com expressões regulares e do conforto oferecido.

A interface foi idealizada para ser simples e intuitiva, por isso optamos pelo básico funcional. Quando o usuário digita seu código na caixa de texto e clica em enviar, a página faz uma requisição para o FastAPI que recebe o código fonte, instancia o interpretador, executa o código e retorna o resultado.

4 GRAMÁTICA

A gramática original da linguagem foi expandida a partir da versão mais recente. Foram adicionados novos recursos para melhor incorporar os exemplos disponibilizados para os teste do interpretador, além de algumas alterações que permitiram uma implementação mais confortável de tais recursos. Foram criadas novas produções para suportar *list comprehension* e *slicing*, além de estruturas de dados mais robustas para facilitar o trabalho com redes neurais, como listas e dicionários.

```
program      → stmts
stmts        → stmt stmts
              | ε
stmt         → compound_stmt
              | simple_stmt
simple_stmt   → declaration
              | "return" expression
              | "break"
              | "continue"
compound_stmt → function_stmt
              | if_stmt
              | for_stmt
              | while_stmt
              | seq_stmt
              | par_stmt
              | channel_stmt
declaration  → var ID ":" TYPE declaration'
declaration' → "=" expression
              | ε
assignment   → assignable "=" expression
              | call
assignable   → ID
              | array
expression   → assignment
              | logic_or
block        → "{" stmts "}"
function_stmt → "func" ID "(" params ")" "->" TYPE block
params       → param params'
params'      → "," param params'
              | ε
param        → ID ":" TYPE default
default      → "=" expression
              | ε
if_stmt      → "if" "(" expression ")" block else_block
```

```

else_block      → "else" block
                | ε
while_stmt      → "while" "(" expression ")" block
for_stmt        → "for" "(" var ID: TYPE in expression ")" block
seq_stmt        → "seq" block
par_stmt        → "par" block
channel_stmt    → s_channel_stmt
                | c_channel_stmt
s_channel_stmt  → "s_channel" ID "{" ID, STRING, STRING, NUMBER "}"
c_channel_stmt  → "c_channel" ID "{" STRING, NUMBER "}"
logic_or        → logic_and logic_or'
logic_or'       → "||" logic_and logic_or'
                | ε
logic_and       → eq logic_and'
logic_and'      → "&&" eq logic_and'
                | ε
eq              → comp eq'
eq'             → ("==" | "!=") comp eq'
                | ε
comp            → sum comp'
comp'           → (">" | "<" | ">=" | "<=") sum comp'
                | ε
sum             → term sum'
sum'            → ("+" | "-") term sum'
                | ε
term            → unary term'
term'           → ("*" | "/" | "%") unary term'
                | ε
unary           → ("!" | "-") unary
                | primary
primary         → "(" logic_or ")"
                | STRING
                | NUMBER
                | "true"
                | "false"
                | list_literal
                | dict_literal
                | call

call            → assignable call'
call'           → "." assignable call'
                | "(" arguments ")" call'
                | ε
array           → ID array'
array'          → index array'
                | ε

```


| | |
|---------------|---------------------------------------------------------------------------------------------|
| index | → "[" index_expr "]" |
| index_expr | → expression slice_expr |
| slice_expr | → expression ":" expression |
| arguments | → args ε |
| args | → expression args' |
| args' | → "," expression args' ε |
| dict_literal | → "{" dict_entries "}" |
| dict_entries | → dict_entry dict_entries' ε |
| dict_entries' | → "," dict_entry dict_entries' ε |
| dict_entry | → STRING ":" expression |
| list_literal | → "[" list_content "]" |
| list_content | → expression list_content' "for" "(" var ID "in" expression ")" "->" expression ε |
| list_content' | → "," expression list_content' ε |

5 ARQUITETURA

A arquitetura geral do projeto foi concebida usando técnicas de engenharia de software para garantir um desenvolvimento sistemático e consistente. Foram elaborados diagramas para ilustrar a estrutura geral do projeto e as interações entre seus componentes principais, a notação da UML foi escolhida para uma melhor organização.

- **Diagramas de casos de uso:** Foram elaborados diagramas para demonstrar as interações do usuário tanto com o interpretador quanto com a interface

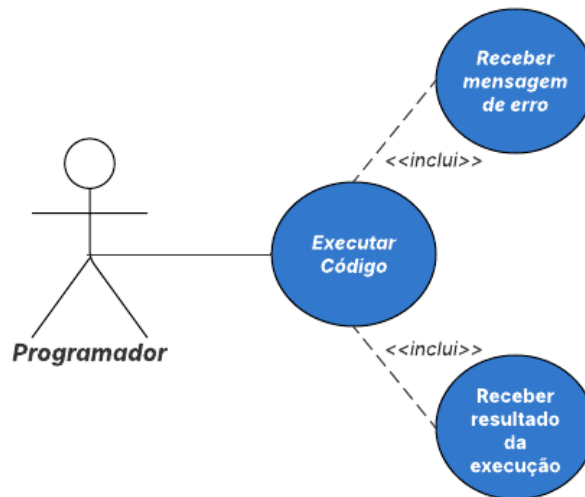


Figura 1: Diagrama de casos de uso Usuário-Interpretador.

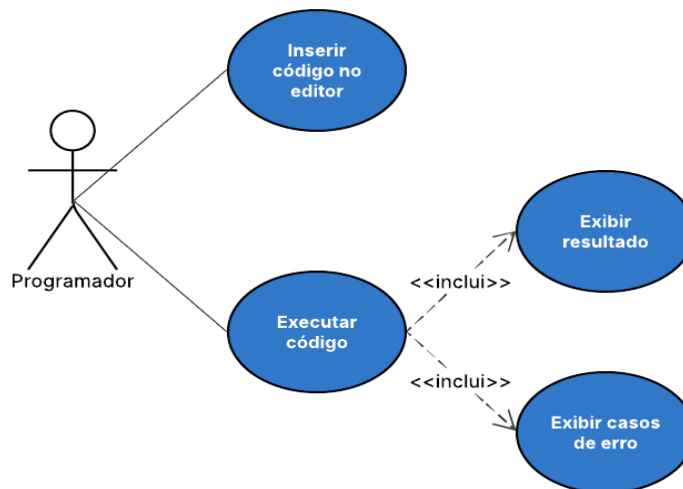


Figura 2: Diagrama de caso de uso Usuário-Interface.

- **Diagrama de componentes:** Demonstra a estrutura modular do sistema, as dependências de cada módulo e suas interações.

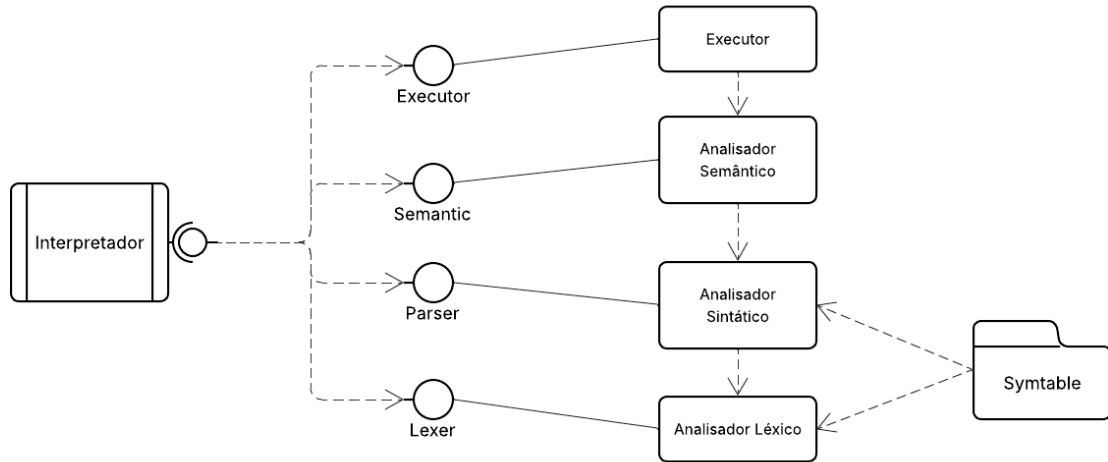


Figura 3: Diagrama de componentes.

- **Diagrama de classes**

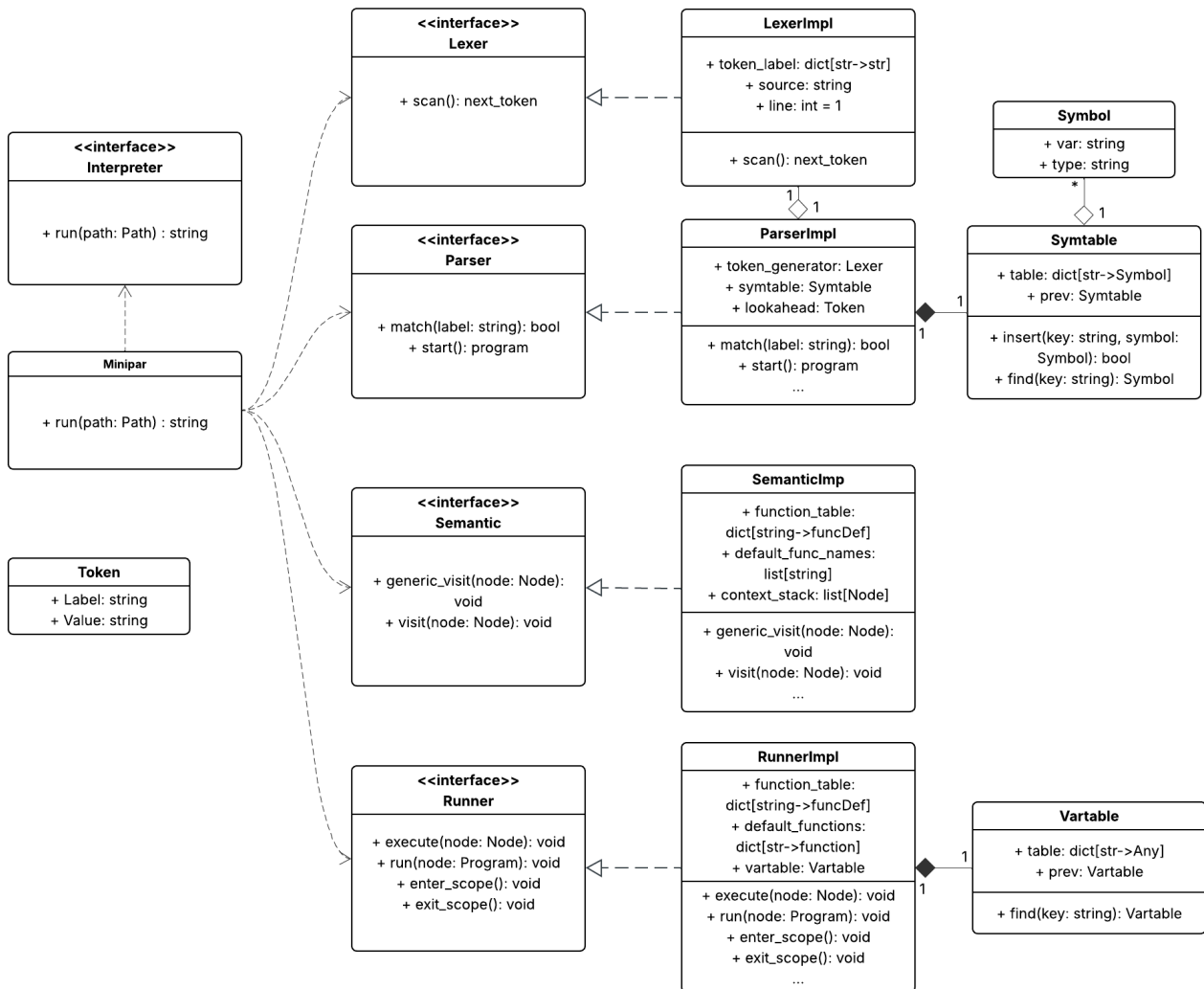


Figura 4: Diagrama de classes.

6 PSEUDOCÓDIGOS

6.1 Analisador Léxico

```
interface Lexer:
    método scan()      .
                        .
classe LexerImpl implementa Lexer:
    source: string
    line: integer
    token_labels: dict[string, string]

    método inicialização:
        Adiciona tipos e valores em token_labels
        Adiciona palavras reservadas em token_labels

    método scan():
        compiled_re = compila_regex(TOKEN_REGEX)
        para cada match em compiled_re.data():
            token_tipo = match.grupo
            token_valor = match.valor
            se token_tipo é espaço em branco ou comentário:
                continue
            se token_tipo é bloco de comentário:
                line += número total de linhas no comentário
            se token_tipo é quebra de linha:
                line += 1
                continue
            se token_tipo é nome:
                token_tipo = token_labels.get(valor, "ID")
            se token_tipo é string:
                remove as aspas duplas
            se token_tipo é outro:
                token_tipo = token_labels.get(token_valor, token_valor)

        gera << Token(token_tipo, token_valor), linha >>
```

6.2 Analisador Sintático

```
interface Parser:
    método match(token: string)
    método start()

classe ParserImpl implementa Parser:
    método inicialização(lexer: Lexer):
        lexer = lexer.scan()
        lookahead, linha = next_token(lexer)
        symtable = Symtable()
        para cada função da biblioteca padrão da linguagem:
            Adiciona essas funções na symtable com a label "FUNC"

    método match(label: string):
        se label == lookahead.label:
            lookahead, linha = next_token(lexer) ou EOF
            retorna true
        se não
            retorna false

    método start():
        retorna program()

    método program():
        retorna Módulo(stmts())

    método stmts()
        body: Body
        enquanto lookahead for uma declaração ou expressão válida:
            body.add(stmt())
        retorna body

    método stmt():
        verifica o lookahead
        caso VAR:
            retorna declaration()
        caso IF:
            if_stmt → "if" "(" expression ")" block else_block
        caso WHILE:
            while_stmt → "while" "(" expression ")" block
        caso FOR:
            for_stmt → "for" "(" var ID: TYPE in expression ")" block
        caso FUNC:
            function_stmt → "func" ID "(" params ")" "->" TYPE block
        caso RETURN:
            match("RETURN")
            expr = logic_or()
            retorna expr
        caso BREAK:
            match("BREAK")
            retorna Break()
        caso CONTINUE:
            match("CONTINUE")
            retorna Continue()
        caso SEQ:
            match("SEQ")
            retorna Seq(block())
```

```

    caso PAR:
        match("PAR")
        retorna Par(block())
    caso C_CHANNEL:
        c_channel_stmt → "c_channel" ID "{" STRING, NUMBER "}"
    caso S_CHANNEL:
        s_channel_stmt → "s_channel" ID "{" ID, STRING, STRING,
NUMBER "}"

    caso nenhum:
        retorna expression()

método params():
    params → param params'
    params' → "," param params' | vazio

método param():
    param → ID ":" TYPE default
    default → "=" expression

método block():
    block → "{" stmts "}"

método declaration():
    declaration → var ID ":" TYPE declaration'
    declaration' → "=" expression

métodos match_id e reference_id // não sei se precisa colocar

método expression():
    expression → assignment | logic_or

método logic_or():
    logic_or → logic_and logic_or'
    logic_or' → "||" logic_and logic_or' | vazio

método logic_and():
    logic_and → equality logic_and'
    logic_and' → "&&" eq logic_and' | vazio

método equality():
    eq → comp eq'
    eq' → ("==" | "!=") comp eq' | vazio

método comp():
    comp → sum comp'
    comp' → (">" | "<" | ">=" | "<=") sum comp' | vazio

método sum():
    sum → term sum'
    sum' → ("+" | "-") term sum' | vazio

método term():
    term → unary term'
    term' → ("*" | "/" | "%") unary term' | vazio

método unary():

```

```

        unary → ("!" | "-") unary | primary

método primary():
    primary → "(" logic_or ")"
            | STRING
            | NUMBER
            | "true"
            | "false"
            | list_literal
            | dict_literal
            | call

método call():
    call → assignable call'
    call' → "." assignable call'
           | "(" arguments ")" call' | vazio

método index():
    index → "[" index_expr "]"

método index_expr():
    index_expr → expression
               | slice_expr

método args():
    arguments → args | vazio
    args → expression args'
    args' → "," expression args' | vazio

método list_literal():
    list_literal → "[" list_content "]"
    list_content → expression list_content'
                  | "for" "(" var ID "in" expression ")" "->"

expression
    list_content' → "," expression list_content' | vazio

método comprehension():
    comprehension -> "[" "for" "(" "var" ID "in" expression ")" "->"
expression "]"

método dict_literal():
    dict_literal → "{" dict_entries "}"
    dict_entries → dict_entry dict_entries' | vazio
    dict_entries' → "," dict_entry dict_entries' | vazio
    dict_entry → STRING ":" expression

```

6.3 Tabela de Símbolos e Variáveis

```

classe symbol:
    var: string
    type: string

classe SymTable:
    table: dict[str, Symbol]
    prev: Symtable

método insert(string: str, symbol: Symbol):
    se table.get(string):
        retorne Falso
    table[string] = symbol
    retorne True

```

```

método find(string: str):
    st = self
    enquanto st:
        valor = table.get(string)
        se valor não é nulo:
            st = st.prev
            continue
        retorne valor

classe VarTable:
    table: dict[str, Any]
    prev: VarTable

    método find(string: str):
        st = self
        enquanto st:
            valor = table.get(string)
            se valor não é nulo:
                st = st.prev
                continue
            retorne st

```

6.4 Analisador Semântico

```

interface Semantic:
    método visit(node: Node)
    método generic_visit(node: Node)

classe SemanticImpl implementa Semantic:
    método inicialização():
        context_stack = []
        function_table = {}

    método visit(node: Node):
        meth_name = "visit_" + tipo(node)
        visitor = get_method(classe, meth_name, default=generic_visit)
        retorna visitor(node)

    método generic_visit(node: Node):
        context_stack.add(node)
        Para cada atributo de node:
            se é uma lista:
                para cada elemento da lista:
                    visit(elemento)
            se é um Node:
                visit(atributo)
        context_stack.pop()

    método visit_Assign(node: Assign):
        se node não é um ID ou Access:
            Erro: Atribuição deve ser feita para uma variável.
        left_type = visit(node.left)
        right_type = visit(node.right)

        se (node.left é um Access ou
            left_type == 'LIST' ou

```



```

        node.right é um Arithmetic):
            retorna

    if left_type != right_type
        Erro de Tipagem

método visit_Declaration(node: Declaration):
    se node.left não é um ID:
        Erro: Atribuição deve ser feita para uma variável

    se node.right não é None:
        left_type = visit(node.left)
        right_type = visit(node.right)

    se node.left é um Access:
        return

    se node.right é um Access ou ArrayLiteral ou Comprehension ou
Arithmetic:
        return

    se left_type != right_type:
        Erro de Tipagem

método visit_Return(node: Return):
    function = Primeiro node que é um FuncDef em context_stack

    se function é None:
        Erro: declaração de retorno fora de uma função.

    expr_type = visit(node.expr)
    se expr_type != function.return_type:
        Erro de Tipagem

método visit_Break(node: Break):
    se algum dos contextos em self.context não é um While ou For:
        Erro: declaração de break fora de um laço.

método visit_Continue(node: Continue):
    se algum dos contextos em self.context não é um While ou For:
        Erro: declaração de continue fora de um laço.

método visit_FuncDef(node: FuncDef):
    se algum dos contextos em self.context é um If ou While ou For:
        Proibido criar funções dentro de escopos locais.

    se node.name não está em function_table:
        function_table.add(node)

    generic_visit(node)

método visit_Block(block: Body):
    para cada node em block:
        visit(node)

método visit_If(node: If):
    cond_type = visit(node.condition)

```

```

        se condition_type != "BOOL":
            Erro de Tipagem

        context_stack.add(node)
        visit_Block(node.body)
        se existir node.else_stmt
            visit_Block(node.else_stmt)
        self.context_stack.pop()

método visit_While(node: While):
    Similar ao If
    - Checa se condição é BOOL
    - Visita o bloco

método visit_For(node: While):
    Similar ao If
    - Checa se o iterável é LIST ou DICT
    - Visita o bloco

método visit_Par(node: Par):
    se algum dos nodes em node.body não for Call:
        Erro: Apenas chamadas de função são permitidas.

método visit_CChannel(node: CChannel):
    host_type = visit(node.host)
    port_type = visit(node.port)

    if host_type != 'STRING':
        Erro de Tipagem

    if port_type != 'NUMBER':
        Erro de Tipagem:

método visit_SChannel(node: SChannel):
    Similar ao CChannel, mas adiciona checagem para:
    - node.description deve ser uma STRING
    - retorno de function deve ser uma STRING

método visit_Slice(node: Slice):
    // Segue a mesma lógica dos outros
    node.initial e node.final devem ser um NUMBER
    retorna node.type

método visit_Comprehention(node: Comprehention):
    node.iterable deve ser um 'LIST' ou 'DICT'
    retorna node.type

método visit_Access(node: Access):
    node_type = visit(node.id)
    node_type deve ser 'LIST' ou 'DICT' ou 'STRING'

    expr = visit(node.expr)
    se node.id é uma 'STRING' ou 'LIST'
        expr_type deve ser 'NUMBER'
    senão
        expr_type deve ser 'STRING'

    retorna node_type

```

```

método visit_ArrayLiteral(node: ArrayLiteral):
    Todos os elementos de node devem ser do mesmo tipo
    retorna esse tipo

método visit_DictLiteral(node: DictLiteral):
    return node.type

método visit_Call(node: Call):
    name = node.function_name

    se name não está em function_table nem em DEFAULT_FUNCTIONS:
        Erro: a função não está definida

    if name in DEFAULT_FUNCTIONS:
        retorna DEFAULT_FUNCTIONS[name]

    function = function_table.get(name)

    para cada argumento em function:
        visit(argumento)

    if n_argumentos < n_parametros_nao_default:
        Erro: Função esperava mais parâmetros

    retorna function.return_type

método visit_Constant(node: Constant):
    retorna node.type

método visit_ID(node: ID):
    retorna node.type

método visit_Logical(node: Logical):
    // visita está implícita
    node.left e node.right devem ser do tipo 'BOOL'

    retorna 'BOOL'

método visit_Relational(node: Relational):
    // visita está implícita
    se node.left ou node.right são Access:
        retorna 'BOOL'

    se node.left e node.right forem de tipos diferentes:
        Erro de Tipagem

    retorna 'BOOL'

método visit_Arithmetic(node: Arithmetic):
    // visita está implícita
    se node.left ou node.right são Access:
        return

    se node.token == "+":
        node.left e node.right devem ter o mesmo tipo
    else
        node.left e node.right devem ser do tipo 'NUMBER'

```

```

        retorna node.left.type

método visit_Unary(node: Unary):
    // visita está implícita
    se node.token == '-':
        node.expr deve ser do tipo 'NUMBER'

    se node.token == '!':
        node.expr deve ser do tipo 'BOOL'

    return node.expr.type

```

6.5 Executor de código

```

interface Runner:
    método run(node: Program):
    método execute(node: Node):
    método enter_scope():
    método exit_scope():

classe RunnerImpl implementa Runner:
    var_table: VarTable
    func_table: dict[str, FuncDef]
    DEFAULT_FUNCTIONS: dict[str, function]

    método inicialização():
        var_table = VarTable()
        func_table = {}

    método execute(node: Node):
        method_name = recupera nome do método
        method = recupera o método cujo nome é method_name para ser chamado

        se method:
            retorne method(node)
        se não:
            gera um erro de método não implementado

    método enter_scope():
        cria novo escopo e atribui à variável local

    método exit_scope():
        sai do escopo atual e retorna para a tabela anterior

    método exec_Declaration(node: Declaration):
        rvalue = resultado da execução do lado direito do nodo da árvore
        var_name = nome da variável declarada
        coloca var_name na tabela de variáveis e atribui rvalue como valor

    método exec_Assign(node: Assign):
        rvalue = resultado da execução do lado direito do nodo da árvore
        var_name = nome da variável declarada
        lvalue_table = resultado da busca pela variável nos escopos existentes
        se lvalue_table:
            lvalue_table[var_name] = rvalue

```

```

        se não
            cria var_name no escopo local e atribui rvalue como valor

método exec_Func_Def(node: FuncDef):
    se o nodo não existe na tabela de funções:
        adiciona na tabela

método exec_Constant(node: Constant):
    match tipo do node:
        caso 'STRING':
            retorne token.value
        caso 'NUMBER':
            retorne eval(token)
        caso 'BOOL':
            retorne bool
        caso nenhum:
            retorne token.value

método exec_ID(node: ID):
    var_name = token
    lvalue_table = var_table[var_name]
    se lvalue_table:
        retorne nome da variável na tabela
    se não:
        gera um erro de variável não definida

método exec_Access(node: Access):
    index = resultado da execução da expressão contida no node
    var_name = token.value
    lvalue_table = find(var_name)
    se lvalue_table:
        retorne lvalue_table[var_name][index]
    se não:
        gera um erro de variável não definida

método exec_logical(node: Logical):
    left = node.left

    match token.value:
        caso '&&':
            se left:
                retorne node.right
            se não:
                retorne left
        caso '||':
            right = node.right
            retorne left ou right
        caso nenhum:
            retorne

método exec_Relational():
    #to do

método exec_Arithmetic():
    #to do

método exec_Unary(node: Unary):
    expr = node.exp
    se expr é nulo:

```

```

        retorne
    match token.value
        caso '!':
            retorne não expr
        caso '-':
            retorne expr * (-1)
        caso nenhum:
            retorne

método exec_ArrayLiteral(node: ArrayLiteral):
    computed_values = []
    para cada item em node.values:
        computed_values.append(execute(value))
    retorne computed_values

método DictLiteral(node: DictLiteral):
    computed_values = {}
    para cada chave e valor em node.items():
        computed_values recebe mapeamento chave e valor
    retorne computed_values

método exec_Call(node: Call):
    nome = node.oper
    se nome está em DEFAULT_FUNCTIONS:
        args = [argumentos em node.args]
        retorne DEFAULT_FUNCTIONS[nome](*args)

    func = func_table[nome]

    se não func:
        gera um erro de função inexistente

    enter_scope()
    para cada parâmetro no conjunto de parâmetros da função chamada
        executar a função com as variáveis presentes na tabela

    retorna resultado da função

método exec_Return(node: Return):
    ReturnInterruption(node.expr)

método exec_Break(node: Break):
    BreakInterruption

método exec_Continue(node: Continue):
    ContinueInterruption

método exec_Block(block: Body):
    ret = none
    para cada instrução em block:
        execute(instrução)
    retorne none

método exec_Comprehension(node: Comprehension):
    result = []
    iterable = node.iterable
    para cada valor em iterable:
        enter_scope()

```

```

        var_table.table[iterator.left.name] = valor
        result.append(execute(node.expr))
        exit_scope()
    retorne result

método exec_For(node: For):
    iterable = node.iterable
    para cada valor em iterable:
        enter_scope()
        var_table.table[iterator.left.name] = valor
        result.append(execute(node.expr))
        exit_scope()

método exec_While(node: While):
    temp = execute(node.condition)
    enquanto temp:
        enter_scope()
        exec_block(node.body)
        temp = execute(node.condition)
    caso ContinueInterruption:
        continue
    caso BreakInterruption:
        break
    exit_scope()

método exec_If(node: If):
    se execute(node.condition):
        enter_scope()
        exec_block(node.body)
        exit_scope()
    se não, se node.else_stmt for diferente de none:
        enter_scope()
        exec_block(node.else_stmt)
        exit_scope()

```

7 TESTES

7.1 Neurônio

Código

```
# Código Simples de um Neuronio
```

```
var input_val: number = 1
var output_desire: number = 0

var input_weight: number = 0.5
var learning_rate: number = 0.01
```

```
# Função de Ativação
```

```
func activation(sum: number) -> number{
    if (sum >= 0) {
        return 1
    }
    else {
        return 0
    }
}
```

```
print("Entrada: ", input_val, " Desejado: ", output_desire)
```

```
# Inicializar erro
```

```
var error: number = 1000.0
var iteration: number = 0
var bias: number = 1
var bias_weight: number = 0.5
```

```
while (error != 0) {
    iteration = iteration + 1
    print("#### Iteração: ", iteration)
    print("Peso: ", input_weight)

    var sum_val: number = (input_val * input_weight) + (bias * bias_weight)

    var output: number = activation(sum_val)
    print("Saída: ", output)

    error = output_desire - output
    print("Erro: ", error)

    if (error != 0) {
        input_weight = input_weight + (learning_rate * input_val * error)
        print("Peso do bias: ", bias_weight)
        bias_weight = bias_weight + (learning_rate * bias * error)
    }

    print("Parabéns!!! A Rede de um Neurônio Aprendeu")
    print("Valor desejado: ", output_desire)
}
```


Output:

```
(outras iterações ...)
#### Iteração: 45
Peso: 0.05999999999999997
Saída: 1
Erro: -1
Peso do bias: 0.05999999999999997
Parabéns!!! A Rede de um Neurônio Aprendeu
Valor desejado: 0
#### Iteração: 46
Peso: 0.04999999999999997
Saída: 1
Erro: -1
Peso do bias: 0.04999999999999997
Parabéns!!! A Rede de um Neurônio Aprendeu
Valor desejado: 0
#### Iteração: 47
Peso: 0.0399999999999999696
Saída: 1
Erro: -1
Peso do bias: 0.0399999999999999696
Parabéns!!! A Rede de um Neurônio Aprendeu
Valor desejado: 0
#### Iteração: 48
Peso: 0.0299999999999999694
Saída: 1
Erro: -1
Peso do bias: 0.0299999999999999694
Parabéns!!! A Rede de um Neurônio Aprendeu
Valor desejado: 0
#### Iteração: 49
Peso: 0.019999999999999969
Saída: 1
Erro: -1
Peso do bias: 0.019999999999999969
Parabéns!!! A Rede de um Neurônio Aprendeu
Valor desejado: 0
#### Iteração: 50
Peso: 0.0099999999999999691
Saída: 1
Erro: -1
Peso do bias: 0.0099999999999999691
Parabéns!!! A Rede de um Neurônio Aprendeu
Valor desejado: 0
#### Iteração: 51
Peso: -3.0878077872387166e-16
Saída: 0
Erro: 0
Parabéns!!! A Rede de um Neurônio Aprendeu
Valor desejado: 0
```

7.2 Quicksort

código

```
# Função para implementar o Quicksort
func quicksort(array: list) -> list {
    if (len(array) <= 1) {
        return array
    }
    else {
        var pivot: any = array[0] # Escolhe o primeiro elemento como pivô

        # Elementos menores ou iguais ao pivô
        var menores: list = []
        for (var x: any in array[1:]) {
            if (x <= pivot) {
                menores.append(x)
            }
        }

        # Elementos maiores ou iguais ao pivô
        var maiores: list = []
        for (var x: any in array[1:]) {
            if (x > pivot) {
                maiores.append(x)
            }
        }

        return quicksort(menores) + [pivot] + quicksort(maiores)
    }
}

# Função para exibir interface simples de texto
func exibir_interface() -> void {
    print("==== Ordenação com Quicksort ====")
    print("Insira os elementos do vetor separados por espaço:")

    # Leitura do vetor
    var input_numbers: string = input()
    var numbers: list = input_numbers.split(" ")
    var numbers_strip: list = [for (var n: string in numbers) -> n.strip()]

    var array: list = [for (var n: string in numbers) -> n.to_number()]
    print("Vetor original: ", array)

    # Aplicar o Quicksort
    var vetor_ordenado: list = quicksort(array)
    print("Vetor ordenado: ", vetor_ordenado)
}

exibir_interface()
```

output:

```
==== Ordenação com Quicksort ====
Insira os elementos do vetor separados por espaço:
9 6 4 8 6 3 9 7 8 5 0 6
Vetor original:  [9, 6, 4, 8, 6, 3, 9, 7, 8, 5, 0, 6]
Vetor ordenado:  [0, 3, 4, 5, 6, 6, 6, 7, 8, 8, 9, 9]
```

7.3 Rede Neural

código

```
# Este código cria uma rede neural com uma camada oculta de três neurônios
# e uma camada de saída com um neurônio, utilizando a função de ativação sigmóide.
# Ele treina a rede para aprender a função XOR usando feedforward e
backpropagation.
# Todos os cálculos são realizados manualmente, sem o uso de bibliotecas externas.

# Função de ativação sigmóide
func sigmoid(x: number) -> number {
    return 1 / (1 + exp(-x))
}

# Derivada da função de ativação sigmóide
func sigmoid_derivative(x: number) -> number {
    return x * (1 - x)
}

# Dados de entrada (função XOR)
var inputs: list = [[0, 0], [0, 1], [1, 0], [1, 1]]
# Saídas desejadas (função XOR)
var outputs: list = [0, 1, 1, 0]

var weights_input_hidden: list = []
for (var i: number in range(2)){
    var linha: list = []
    for (var j: number in range(3)){
        linha.append(random())
    }
    weights_input_hidden.append(linha)
}

var weights_hidden_output: list = [for (var _ : any in range(3)) -> random()]
var bias_hidden: list = [for (var _ : any in range(3)) -> random()]
var bias_output: number = random()

# Taxa de aprendizado
var learning_rate: number = 0.2

for (var epoch: number in range(20000)) {
    for (var i: number in range(len(inputs))) {

        # Fase de feedforward
        var hidden_layer_input: list = []
        for (var k: number in range(3)) {
            for (var j: number in range(2)) {
                hidden_layer_input.append(inputs[i][j]*[j][k])
            }
        }

        var hidden_layer_output: list = [for (var i: number in range(3)) ->
sigmoid(sum(hidden_layer_input[i * 2 : (i + 1) * 2]) + bias_hidden[i])]
        var output_layer_input: number = sum([for (var j: number in range(3))
-> hidden_layer_output[j] * weights_hidden_output[j]]) + bias_output
        var predicted_output: number = sigmoid(output_layer_input)
```

```

        # Cálculo do erro
        var error: number = outputs[i] - predicted_output

        # Fase de backpropagation
        var d_predicted_output: number = error *
sigmoid_derivative(predicted_output)
        var d_hidden_layer: list = [for (var j: number in range(3)) ->
d_predicted_output * weights_hidden_output[j] *
sigmoid_derivative(hidden_layer_output[j])]

        # Atualização dos pesos e bias
        weights_hidden_output = [for (var j: number in range(3)) ->
weights_hidden_output[j] + hidden_layer_output[j] * d_predicted_output *
learning_rate]

        bias_output = bias_output + d_predicted_output * learning_rate

        for (var j: number in range(2)){
            for (var k: number in range(3)){
                weights_input_hidden[j][k] = weights_input_hidden[j][k] +
inputs[i][j] * d_hidden_layer[k] * learning_rate
                bias_hidden[k] = bias_hidden[k] + d_hidden_layer[k] *
learning_rate
            }
        }
    }
}

# Testando a rede neural treinada
for (var i: number in range(len(inputs))) {
    var hidden_layer_input: list = []
    for (var k: number in range(3)) {
        for (var j: number in range(2)) {
            hidden_layer_input.append(inputs[i][j] *
weights_input_hidden[j][k])
        }
    }

    var hidden_layer_output: list = [for (var i: number in range(3)) ->
sigmoid(sum(hidden_layer_input[i * 2 : (i + 1) * 2]) + bias_hidden[i])]
    var output_layer_input: number = sum([for (var j: number in range(3)) ->
hidden_layer_output[j] * weights_hidden_output[j]]) + bias_output
    var predicted_output: number = sigmoid(output_layer_input)
    print("Input:", inputs[i], ", Predicted Output: ", predicted_output)
}

```

Output:

```

Input: [0, 0] , Predicted Output: 0.021364823337103274
Input: [0, 1] , Predicted Output: 0.9834300975311268
Input: [1, 0] , Predicted Output: 0.9834300565965598
Input: [1, 1] , Predicted Output: 0.015434553901951264

```

7.4 Recomendação

código

```
func read_data() -> dict {
    # Função para ler os dados de avaliação dos usuários para produtos
    # Os dados são representados como um dicionário de dicionários
    var user_ratings: dict = {
        "user1": {"product1": 5, "product2": 3, "product3": 4},
        "user2": {"product1": 4, "product2": 2, "product3": 5},
        "user3": {"product2": 5, "product3": 3},
        "user4": {"product1": 3, "product3": 4},
    }
    return user_ratings
}

func calculate_similarity(user1_ratings: dict, user2_ratings: dict) -> number {
    # Calcula a similaridade entre dois usuários usando similaridade de cosseno
    var common_products: list = intersection(user1_ratings.keys(),
user2_ratings.keys())

    if (!common_products) {
        return 0
    }

    var sum1_sq: number = sum([for (var product: string in common_products) ->
pow(user1_ratings[product], 2)])
    var sum2_sq: number = sum([for (var product: string in common_products) ->
pow(user2_ratings[product], 2)])

    var product_sum: number = sum([for (var product: string in common_products)
-> user1_ratings[product] * user2_ratings[product]])

    var numerator: number = product_sum
    var denominator: number = sqrt(sum1_sq) * sqrt(sum2_sq)

    if (denominator == 0) {
        return 0
    }

    return numerator / denominator
}

func get_recommendations(user_ratings: dict, target_user: string) -> list {
    # Obtém recomendações de produtos para o usuário-alvo

    var totals: dict = {}
    var sim_sums: dict = {}

    for (var other_user: string in user_ratings) {
        if (other_user == target_user) {
            continue
        }

        var similarity: number = calculate_similarity(user_ratings[target_user],
user_ratings[other_user])

        if (similarity <= 0) {
            continue
        }
    }
}
```

```

    }
    # Debug: Print user similarities
    print("Similaridade entre ", target_user, " e ", other_user, ": ",
similarity)

    for (var other: list in user_ratings[other_user].items()) {
        var product: string = other[0]
        var rating: number = other[1]

        if (!contains(user_ratings[target_user], product)) {
            if (!contains(totals, product)) {
                totals[product] = 0
            }
            totals[product] = totals[product] + rating * similarity

            if (!contains(sim_sums, product)){
                sim_sums[product] = 0
            }

            sim_sums[product] = sim_sums[product] + similarity
        }
    }
}

var rankings: list = []
for (var rank: list in totals.items()){
    var product: string = rank[0]
    var total: number = rank[1]

    if (sim_sums[product] != 0){
        rankings.append([total / sim_sums[product], product])
    }
}
sort(rankings, true)

var recommendations: list = [for(var rank: list in rankings) -> rank[1]]
return recommendations
}

func main() -> void {
    var user_ratings: dict = read_data()
    while (true) {
        print("Bem-vindo ao Sistema de Recomendação de E-commerce")

        var users: list = [for (var user: list in user_ratings) -> user]
        print("Usuários disponíveis:", users)
        var target_user: string = input("Digite o usuário para o qual deseja
recomendações (ou 'sair' para finalizar): ")
        target_user = target_user.strip()

        if (target_user.lower() == "sair") {
            print("Saindo do sistema de recomendação.")
            break
        }
        if (!contains(user_ratings, target_user)) {
            print("Usuário não encontrado. Tente novamente.")
            continue
        }
    }
}

```

```

        var recommendations: list = get_recommendations(user_ratings,
target_user)
        if (recommendations) {
            print("Recomendações para ", target_user, ": ",
recommendations)
        }
        else {
            print("Nenhuma recomendação disponível para ", target_user,
".")
        }
    }
}

main()

```

Output:

```

Bem-vindo ao Sistema de Recomendação de E-commerce
Usuários disponíveis: ['user1', 'user2', 'user3', 'user4']
Digite o usuário para o qual deseja recomendações (ou 'sair' para finalizar): user1
Similaridade entre user1 e user2 : 0.9697651491183029
Similaridade entre user1 e user3 : 0.9260923597695477
Similaridade entre user1 e user4 : 0.9682773237093576
Nenhuma recomendação disponível para user1 .
Bem-vindo ao Sistema de Recomendação de E-commerce
Usuários disponíveis: ['user1', 'user2', 'user3', 'user4']
Digite o usuário para o qual deseja recomendações (ou 'sair' para finalizar): user2
Similaridade entre user2 e user1 : 0.9697651491183029
Similaridade entre user2 e user3 : 0.7961621941231025
Similaridade entre user2 e user4 : 0.9995120760870788
Nenhuma recomendação disponível para user2 .
Bem-vindo ao Sistema de Recomendação de E-commerce
Usuários disponíveis: ['user1', 'user2', 'user3', 'user4']
Digite o usuário para o qual deseja recomendações (ou 'sair' para finalizar): user3
Similaridade entre user3 e user1 : 0.9260923597695477
Similaridade entre user3 e user2 : 0.7961621941231025
Similaridade entre user3 e user4 : 1.0
Recomendações para user3 : ['product1']
Bem-vindo ao Sistema de Recomendação de E-commerce
Usuários disponíveis: ['user1', 'user2', 'user3', 'user4']
Digite o usuário para o qual deseja recomendações (ou 'sair' para finalizar): user4
Similaridade entre user4 e user1 : 0.9682773237093576
Similaridade entre user4 e user2 : 0.9995120760870788
Similaridade entre user4 e user3 : 1.0
Recomendações para user4 : ['product2']
Bem-vindo ao Sistema de Recomendação de E-commerce
Usuários disponíveis: ['user1', 'user2', 'user3', 'user4']

```

7.5 Cliente - Servidor

Código do Cliente

```
c_channel client {"localhost", 8585}

while(true){
    var entrada: string = input("Digite uma expressão: ")
    if (entrada == "exit")
    {
        break
    }
    var ret: string = client.send(entrada)
    print(ret)
}

client.close()
```

Código do Servidor

```
func calc(message: string) -> string{
    var index: number = 0
    var size: number = len(message)
    var result: number = 0
    var operator: string = "+"
    print(message)
    while(index < size)
    {
        if(isalpha(message[index])){return "INVALIDO"}
        if(message[index] == " "){
            index = index + 1
            continue
        }
        if(isnum(message[index])){
            var valor: string = message[index]
            index = index + 1
            while(index < size && isnum(message[index])){
                valor = valor + message[index]
                index = index + 1
            }
            var valor_num: number = to_number(valor)
            if(operator == "+"){result = result + valor_num}
            else{if(operator == "-"){result = result - valor_num}
            else{if(operator == "*"){result = result * valor_num}
            else{if(operator == "/"){result = result / valor_num}
            else{return "INVALIDO"}}}}}
        }else{
            operator = message[index]
            index = index + 1
        }
    }
    return to_string(result)
}

var description: string = "CALCULADORA SIMPLES"
s_channel server {calc, description, "localhost", 8585}
```


Output do Cliente

CALCULADORA SIMPLES

Digite uma expressão: 1 + 2 + 3

6

Digite uma expressão: 7 + 2 * 4 - 1

35

Digite uma expressão: 8 / 2 - 1

3.0

Digite uma expressão:

Output do Servidor

Cliente conectado.

1 + 2 + 3

6

7 + 2 * 4 - 1

35

8 / 2 - 1

3.0

Cliente desconectado.

7.5.1 Servidor em Java com Minipar Cliente

Código do Servidor

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketMinipar {
    public static String calc(String mensagem) {
        int index = 0;
        int tamanho = mensagem.length();
        int resultado = 0;
        char operador = '+';

        System.out.println("Recebido: " + mensagem);

        while (index < tamanho) {
            char c = mensagem.charAt(index);

            if (Character.isAlphabetic(c)) return "INVALIDO";

            if (c == ' ') {
                index++;
                continue;
            }

            if (Character.isDigit(c)) {
                StringBuilder valorStr = new StringBuilder();
                while (index < tamanho && Character.isDigit(mensagem.charAt(index))) {
                    valorStr.append(mensagem.charAt(index));
                    index++;
                }
            }
        }
    }
}
```

```

        int valor = Integer.parseInt(valorStr.toString());

        switch (operador) {
            case '+': resultado += valor; break;
            case '-': resultado -= valor; break;
            case '*': resultado *= valor; break;
            case '/':
                if (valor == 0) return "DIVISAO POR ZERO";
                resultado /= valor; break;
            default: return "INVALIDO";
        }
    } else {
        operador = c;
        index++;
    }
}

return String.valueOf(resultado);
}

public static void main(String[] args) throws IOException {
    String descricao = "CALCULADORA SIMPLES\n";
    ServerSocket server = new ServerSocket(8585);
    try {
        System.out.println("Servidor iniciado na porta 8585...");
        while (true) {
            Socket client = server.accept();
            System.out.println("Cliente conectado: " + client.getInetAddress());
            BufferedReader in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
            PrintWriter out = new PrintWriter(client.getOutputStream(), true);
            out.println(descricao);
            String entrada;
            while ((entrada = in.readLine()) != null) {
                String resposta = calc(entrada);
                out.println(resposta+"\n");
            }
            client.close();
            System.out.println("Cliente desconectado.");
        }
    } catch (IOException e) {
        System.err.println("Erro no servidor: " + e.getMessage());
    } finally {
        server.close();
    }
}
}

```

Output do Servidor

```

Servidor iniciado na porta 8585...
Cliente conectado: /127.0.0.1
Recebido: 3+4
Recebido: 7*2+1
Cliente desconectado.

```

Output do Cliente

CALCULADORA SIMPLES

Digite uma expressão: 3+4

7

Digite uma expressão: 7*2+1

15

Digite uma expressão: exit

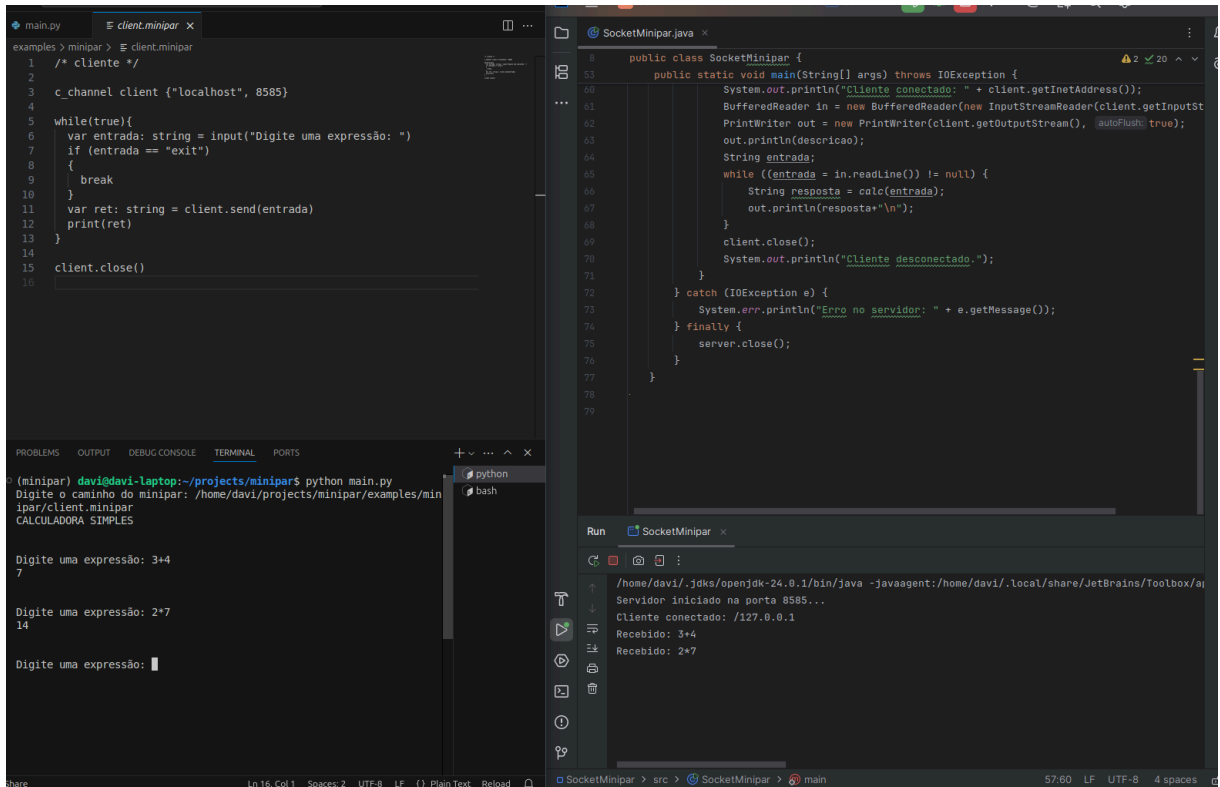


Figura 5: Screenshot de outra execução.

7.6 Execução paralela do Fatorial e Fibonacci

Código

```
/* calcula o fatorial de x a y */
func fatorial(x: number, y: number) -> void
{
    func fat(n: number) -> number{
        var prod: number = 1
        var i: number = 2
        while(i <= n){
            prod = prod * i
            i = i + 1
        }
        return prod
    }

    var i: number = x
    while(i <= y)
    {
        print("Fatorial de:", i, "=", fat(i))
        i = i + 1
        sleep(0.5)
    }
}
```

```

    }
}

/* calcula os n primeiros termos de fibonacci */
func fibonacci(n: number) -> void{
    var a: number = 0
    var b: number = 1
    var count: number = 0

    while (count < n)
    {
        print("Fib:", a)
        var aux: number = a + b
        a = b
        b = aux
        count = count + 1
        sleep(0.5)
    }
}

par{
    fatorial(2, 5)
    fibonacci(10)
}

```

Output

```

Fib: 0
Fatorial de: 2 = 2
Fib: 1
Fatorial de: 3 = 6
Fib: 1
Fatorial de: 4 = 24
Fib: 2
Fatorial de: 5 = 120
Fib: 3
Fib: 5
Fib: 8
Fib: 13
Fib: 21
Fib: 34

```

7.7 Fatorial recursivo

código

```

/* exemplo de recursao */
func fatorial(n: number) -> number
{
    if (n == 0 || n == 1)
    {
        return 1
    }
    else {

```

```

    return n * fatorial(n - 1)
  }
}

print("CALCULA O FATORIAL RECURSIVO")
var valor: number = 10
print("Fatorial: ", fatorial(valor))

```

output

```

CALCULA O FATORIAL RECURSIVO
Fatorial: 3628800

```

7.8 Recomendação 2

```

# Histórico de compras do usuário
var user_purchase_history: list = ["Smartphone", "Jeans", "Micro-ondas", "Ficção"]

# Categorias de produtos e seus itens
var product_categories: dict = {
  "Eletrônicos": ["Smartphone", "Laptop", "Tablet", "Fones de ouvido"],
  "Roupas": ["Camisa", "Jeans", "Jaqueta", "Sapatos"],
  "Eletrodomésticos": [
    "Geladeira",
    "Micro-ondas",
    "Máquina de lavar",
    "Ar condicionado",
  ],
  "Livros": ["Ficção", "Não-ficção", "Ficção científica", "Fantasia"],
}

func index(product_list: list, target_product: string) -> number {
  var product_index: number
  for (var i: number in range(len(product_list))) {
    if (product_list[i] == target_product) {
      product_index = i
      break
    }
  }
  return product_index
}

# Função para codificar o histórico de compras do usuário
func encode_purchase_history(user_purchase_history: list, product_categories: dict)
-> list {
  var all_products: list = []
  for (var category: string in product_categories) {
    for (var product: string in product_categories[category]) {
      all_products.append(product)
    }
  }
  var encoded_history: list = [for (var i: number in range(len(all_products))) ->
0]

  for (var product: string in user_purchase_history) {
    if (contains(all_products, product)) {
      encoded_history[index(all_products, product)] = 1
    }
  }
}

```

```

    }
  }
  return encoded_history
}

# Função para criar a rede neural
func create_neural_network(input_size: number, hidden_size: number , output_size:
number) -> list {
  var W1: list = []
  for (var i: number in range(input_size)) {
    var row: list = []
    for (var j: number in range(hidden_size)) {
      row.append(0.5)
    }
    W1.append(row)
  }
  var b1: list = [for(var i: number in range(hidden_size)) -> 0.5]
  var W2: list = []
  for (var i: number in range(hidden_size)) {
    var row: list = []
    for (var j: number in range(output_size)) {
      row.append(0.5)
    }
    W2.append(row)
  }
  var b2: list = [for(var i: number in range(output_size)) -> 0.5]

  return [W1, b1, W2, b2]
}

func max(initial: number, final: number) -> number {
  if (initial > final) {
    return initial
  } else {
    return final
  }
}

# Função para ativação ReLU
func relu(x: list) -> list {
  var res: list = [for(var i: any in x) -> max(0, i)]
  return res
}

# Função para ativação Sigmoid
func sigmoid(x: list) -> list {
  var res: list = [for(var i: any in x) -> 1 / (1 + exp(-i))]
  return res
}

# Função para a propagação da rede neural
func forward_propagation(X: list, W1: list, b1: list, W2: list, b2: list) -> list {
  var Z1: list = []
  for(var i: number in range(len(b1))){
    var soma: number = sum([for(var j: number in range(len(X))) -> X[j] *
W1[j][i]])
    Z1.append(soma + b1[i])
  }
}

```

```

    var A1: list = relu(Z1)
    var Z2: list = []
    for(var i: number in range(len(b2))){
        var soma: number = sum([for(var j: number in range(len(A1))) -> A1[j] *
W2[j][i]])
        Z2.append(soma + b2[i])
    }

    var A2: list = sigmoid(Z2)
    return A2
}

# Função para recomendar produtos
func recommend_products(user_purchase_history: list, product_categories: dict) ->
list {
    var encoded_history: list = encode_purchase_history(user_purchase_history,
product_categories)

    var input_size: number = len(encoded_history)
    var hidden_size: number = 10 # Tamanho da camada oculta
    var output_size: number = len(encoded_history)

    var res: list = create_neural_network(input_size, hidden_size, output_size)
    var W1: list = res[0]
    var b1: list = res[1]
    var W2: list = res[2]
    var b2: list = res[3]

    var recommendations_encoded: list = forward_propagation(encoded_history, W1,
b1, W2, b2)

    var all_products: list = []
    for (var category: string in product_categories) {
        for (var product: string in product_categories[category]) {
            all_products.append(product)
        }
    }

    var recommendations: list = []
    for (var i: number in range(len(all_products))) {
        if (recommendations_encoded[i] > 0.5 && !contains(user_purchase_history,
all_products[i])) {
            recommendations.append(all_products[i])
        }
    }

    return recommendations
}

var recommendations: list = recommend_products(user_purchase_history,
product_categories)

# Imprimir as recomendações
print("Produtos recomendados para você:")
for (var product: any in recommendations){
    print(product)
}

```

output

Produtos recomendados para você:

Laptop

Tablet

Fones de ouvido

Camisa

Jaqueta

Sapatos

Geladeira

Máquina de lavar

Ar condicionado

Não-ficção

Ficção científica

Fantasia

8 INTERFACE

Nessa seção, foram apresentadas as principais funcionalidades da interface gráfica para o interpretador da linguagem Minipar. As funcionalidades incluem um editor de código com *syntax highlighting*, a execução do código e exibição do output, botão de limpar a tela para digitar novos códigos e função opção para inserir inputs de teclado, que são redirecionados para o `stdin` do programa. As Figuras abaixo mostram essas funcionalidades.



Figura 6: Página do editor.



Figura 7: Inserir código.



Figura 8: Executando o código.



Figura 9: Output.



Figura 10: Input para o stdin.

9 HIGHLIGHT SYNTAX

Para facilitar a leitura e o desenvolvimento de programas escritos em Minipar, foi criado um highlight de sintaxe personalizado para o editor Visual Studio Code (VS Code). A gramática foi definida em um arquivo `.json`, onde foram especificados os principais elementos da linguagem, como:

- Palavras-chave (`func`, `var`, `if`, `else`, etc.)
- Tipos de dados (`number`, `string`, `dict`, `list`, etc.)
- Funções nativas (`print`, `input`, `sum`, `contains`, etc.)
- Literais numéricos e strings
- Identificadores
- Comentários de linha (`# ...`) e de bloco (`/* ... */`)
- Operadores (`+`, `-`, `->`, etc.)

Além disso, foi definido um tema de cores básico associado a esses elementos, o que permite que eles sejam destacados de forma visualmente distinta durante a escrita do código. Isso contribui para uma melhor compreensão estrutural dos programas e maior produtividade durante o desenvolvimento. A extensão pode ser empacotada em um arquivo `.vsix` e compartilhada com outros usuários, que podem instalá-la localmente com um simples comando no terminal do VS Code.

10 LINK

<https://github.com/Arthurpmrs/minipar>