Membros do Grupo:

ANTÔNIO ENZO FERREIRA DO NASCIMENTO

RAFAEL NOGUEIRA LEAL REBELO

RAFAEL VAZ ROCHA

1. Introdução

Este documento especifica a linguagem de programação **MiniLua**, uma versão simplificada inspirada na linguagem Lua, desenvolvida para fins didáticos na disciplina de Compiladores.

MiniLua é uma linguagem **imperativa**, com **escopo léxico** e **tipagem estática e explícita**. A sintaxe é similar à de Lua, com blocos delimitados por do ... end, if ... then ... end, function ... end, entre outros.

A linguagem é **case-sensitive**: identificadores como valor, Valor e VALOR são considerados distintos.

Comentário de linha é iniciado por -- e se estende até o fim da linha.
Comentários de bloco são delimitados por --[[e]].

2. Tipos básicos e derivados (I)

2.1. Tipos básicos

MiniLua possui os seguintes tipos básicos:

- **number**
  Tipo numérico de ponto flutuante em precisão dupla.
- **integer**
  Subconjunto de number representando valores inteiros.
  (A implementação pode representar internamente como number, mas semanticamente são inteiros.)
- **boolean**
  Tipo lógico, com dois valores possíveis: true e false.
- **string**
  Sequência de caracteres delimitada por aspas duplas ("texto").
- **nil**
  Representa ausência de valor. Não é utilizado como tipo declarado de

variável, apenas como valor especial (por exemplo, valor padrão de arrays não inicializados).

2.2. Tipos derivados

MiniLua suporta um tipo derivado de coleção:

- **array<T>**
  Estrutura de vetor indexado por inteiros positivos, iniciando em 1.
  Todos os elementos de um array<T> possuem o mesmo tipo T.
  A linguagem não suporta tabelas genéricas como em Lua completa, apenas arrays homogêneos.

Exemplos de tipos derivados:

- array<number>
- array<string>

Exemplos de declaração (ver sintaxe formal em 3.2):

local notas : array<number>;

local nomes : array<string>;

3. Variáveis, constantes e escopo (III)

3.1. Identificadores

Identificadores são usados para nomear variáveis, constantes e funções.

- Devem iniciar com uma letra (a–z, A–Z) ou underscore (_).
- Os demais caracteres podem ser letras, dígitos (0–9) ou underscore.
- A linguagem é **case-sensitive**.
- Identificadores não podem coincidir com palavras reservadas (por exemplo, if, then, end, local, function, and, or, etc.).

3.2. Declaração de variáveis

Todas as variáveis devem ser declaradas com tipo antes de serem utilizadas.
A palavra-chave para declaração de variáveis locais é local.

Sintaxe (informal):

<decl_var> ::= "local" <ident> ":" <tipo> ["=" <expressao>] ";"

Exemplos:

```
local x : integer;
```

```
local y : number = 3.14;

local nome : string = "Joao";

local ativo : boolean = true;

local notas : array<number>;
```

3.2.1. Valores padrão

Na ausência de inicialização explícita:

- integer / number → 0
- boolean → false
- string → "" (string vazia)
- array<T> → nil (não inicializado)

3.3. Declaração de constantes

MiniLua suporta constantes via palavra-chave const. O valor de uma constante deve ser fornecido no momento da declaração e **não pode ser alterado** posteriormente.

Sintaxe (informal):

<decl_const> ::= "const" <ident> ":" <tipo> "=" <expressao> ";"

Exemplos:

```
const PI : number = 3.14159;

const MAX_ALUNOS : integer = 50;

const MSG : string = "Bem-vindo";
```

Qualquer tentativa de atribuição a uma constante deve ser reportada como erro em tempo de compilação.

3.4. Escopo

MiniLua utiliza **escopo léxico** baseado em blocos.

- Variáveis e constantes declaradas com local ou const dentro de um bloco (delimitado por do ... end, if ... end, while ... end, function ... end, etc.) são **visíveis apenas nesse bloco e em seus sub-blocos**.
- Não existem variáveis globais implícitas; todo identificador deve ser declarado em algum bloco visível.

Exemplo:

```
do
    local x : integer = 10;
    if x > 5 then
        local y : integer = 20;
        print(x, y);  -- permitido
    end
    print(x);         -- permitido
    -- print(y);      -- ERRO: y fora de escopo
end

-- print(x);          -- ERRO: x fora de escopo
```

4. Operadores e precedência (II)

4.1. Operadores unários

MiniLua possui os seguintes operadores unários:

- - (menos unário)
    - Aplica-se a operandos de tipo number ou integer.
    - Invertem o sinal do operando.
- not (negação lógica)
    - Aplica-se a operandos de tipo boolean.
    - Retorna true se o operando for false, e false caso contrário.

4.2. Operadores aritméticos binários

- + : soma (number/integer)
- - : subtração
- * : multiplicação
- / : divisão real (resultado sempre number)
- // : divisão inteira (resultado integer)
- % : resto da divisão inteira
- ^ : exponenciação (number)

4.3. Operador de concatenação

- .. : concatenação de strings
    - Ambos os operandos devem ser do tipo string.
    - O resultado é uma string.

4.4. Operadores relacionais

Todos retornam um valor do tipo boolean:

- == : igualdade
- ~= : diferente
- < : menor que
- <= : menor ou igual
- > : maior que
- >= : maior ou igual

Comparações entre number e integer são permitidas, com coerção implícita para number. Comparações entre string seguem ordem lexicográfica.

4.5. Operadores lógicos binários

- and
    - Retorna true se ambos os operandos forem true, caso contrário retorna false.
- or
    - Retorna true se pelo menos um dos operandos for true, caso contrário retorna false.

Ambos esperam operandos de tipo boolean e retornam boolean.

4.6. Precedência e associatividade

Quando múltiplos operadores aparecem em uma mesma expressão:

1. Operadores com **maior precedência** são avaliados primeiro.
2. Operadores com a mesma precedência são avaliados da esquerda para a direita, **exceto** ^, que é associativo à direita.
3. Parênteses (...) podem ser utilizados para alterar a ordem de avaliação padrão.

Tabela de precedência (da **maior** para a **menor**):

Nível	Operador(es)	Aridade	Associatividade	Tipo dos operandos	Tipo do resultado
1	^	binário	direita	number/integer	number
2	- (unário), not	unário	direita	number/integer / boolean	number/integer / boolean
3	*, /, //, %	binário	esquerda	number/integer	number/integer
4	+, -	binário	esquerda	number/integer	number/integer
5	..	binário	direita	string	string
6	==, ~=, <, <=, >, >=	binário	esquerda	number/integer/string	boolean
7	and	binário	esquerda	boolean	boolean
8	or	binário	esquerda	boolean	boolean

5. Declaração de funções (IV)

5.1. Estrutura geral

Funções em MiniLua possuem:

- um tipo de retorno (ou void para ausência de valor),
- um nome,
- uma lista (possivelmente vazia) de parâmetros tipados,
- um corpo formado por declarações locais e comandos.

Funções podem ser **recursivas**. Nesta versão, funções aninhadas (funções declaradas dentro de outras funções) não são suportadas.

5.2. Sintaxe de declaração

Sintaxe (informal):

<decl_func> ::= "function" <tipo_retorno> <ident> "(" [<lista_params>] ")" <bloco> "end"

<tipo_retorno> ::= <tipo> | "void"

<lista_params> ::= <param> { "," <param> }

<param> ::= <ident> ":" <tipo>

<bloco> ::= { <decl_var> | <decl_const> | <comando> }

Todos os parâmetros são passados por valor.

5.2.1. Exemplos

Função com retorno:

```
function number media(n1 : number, n2 : number)
    local m : number;
    m = (n1 + n2) / 2;
    return m;
end
```

Função void:

```
function void mostraMedia(n1 : number, n2 : number)
    local m : number;
    m = (n1 + n2) / 2;
    print("Media: ", m);
end
```

5.3. Comandos de retorno

- Funções com tipo de retorno diferente de void devem conter pelo menos um comando return <expressao>; em todos os caminhos de execução.
- Funções com tipo de retorno void podem conter return; opcionalmente.

Exemplos:

```
return resultado;  -- em função com tipo number
return;            -- em função void
```

6. Funções nativas (V)

MiniLua define um pequeno conjunto de funções nativas (built-ins) para operações de entrada e saída e utilitários básicos.

6.1. Função print

A função print escreve valores na saída padrão (console). Os argumentos são separados por um espaço, e ao final da chamada é inserida uma quebra de linha.

Assinatura conceitual:

function void print(...expressions)

Sintaxe de chamada:

<chamada_print> ::= "print" "(" [<lista_exp>] ")" ";"

<lista_exp> ::= <expressao> { "," <expressao> }

Exemplos:

```
print("Hello", "World");
```

```
print("1 + 1 = ", 1 + 1);
```

```
print(resultado);
```

6.2. Função inputNumber

Lê um número da entrada padrão e retorna um valor number.

function number inputNumber()

Exemplo:

```
print("Digite um numero:");
```

```
local x : number = inputNumber();
```

6.3. Função inputString

Lê uma linha de texto da entrada padrão e retorna um valor string.

function string inputString()

Exemplo:

```
print("Digite seu nome:");
```

```
local nome : string = inputString();
```

6.4. Funções auxiliares

MiniLua inclui também as seguintes funções nativas:

- len(s : string) : integer – retorna o tamanho da string s.
- arrayLength(a : array<T>) : integer – retorna o tamanho do array a.

7. Sentenças de atribuição e controle (VI)

7.1. Atribuição simples

A atribuição utiliza o operador =.

Sintaxe (informal):

<cmd_atribuicao> ::= <ident> "=" <expressao> ";"

Regras semânticas:

- O tipo da expressão à direita deve ser compatível com o tipo da variável à esquerda.
- É permitida a coerção implícita de `integer` para `number`.
- Não é permitida a coerção de `number` para `integer` (perda de precisão), nem entre tipos numéricos e `string` ou `boolean`.

Exemplo:

```
local x : integer = 0;

local y : number;


y = 3.5;

x = 10;
```

7.2. Atribuição em arrays

MiniLua suporta acesso indexado a arrays, com índices inteiros começando em 1.

Sintaxe (informal):

<cmd_atribuicao_array> ::= <ident> "[" <expressao_inteira> "]" "=" <expressao> ";"

Exemplos:

```
local notas : array<number>;
```

```
-- suponha que 'notas' já tenha sido inicializado

notas[1] = 7.5;

notas[2] = 8.0;
```

A expressão usada como índice deve ter tipo integer.
O tipo da expressão à direita deve coincidir com o tipo de elementos do array.

7.3. Bloco Do end

O comando `do` cria um novo escopo léxico explícito. Variáveis declaradas dentro deste bloco não são visíveis fora dele.

Sintaxe (informal):

<cmd_do> ::= "do" <bloco> "end"

Exemplo:

```
local x : integer = 10;

do

  local x : integer = 20;         -- Variável local ao bloco
(sombreamento)

  print(x); -- Imprime 20

end

print(x); -- Imprime 10
```

7.4. Comando condicional if

O comando if permite seleção condicional de blocos de código.

Sintaxe (informal):

<cmd_if> ::= "if" <expressao_booleana> "then" <bloco>

{ "elseif" <expressao_booleana> "then" <bloco> }

["else" <bloco>]

"end"

Exemplo:

```
if x > 0 then
    print("Positivo");
elseif x < 0 then
    print("Negativo");
else
    print("Zero");
end
```

A expressão em if e elseif deve ter tipo boolean.

7.5. Comando de repetição while

O comando while executa um bloco enquanto uma condição booleana for verdadeira.

Sintaxe (informal):

<cmd_while> ::= "while" <expressao_booleana> "do" <bloco> "end"

Exemplo:

```
local i : integer = 1;
while i <= 10 do
    print(i);
    i = i + 1;
end
```

7.6. Comando de repetição for numérico

MiniLua implementa um for numérico com variável de controle inteira.

Sintaxe (informal):

<cmd_for> ::= "for" <ident> "=" <exp_inicio> "," <exp_fim> ["," <exp_passo>] "do" <bloco> "end"

- <exp_inicio>, <exp_fim> e <exp_passo> são expressões de tipo integer.
- Se <exp_passo> não for especificado, assume-se passo 1.

Exemplos:

```
for i = 1, 10 do
    print(i);
end

for i = 10, 1, -1 do
    print(i);
end
```

A variável de controle do for é considerada local ao laço.

7.7. Comando return

O comando return é permitido apenas dentro de funções.

Sintaxe (informal):

<cmd_return_val> ::= "return" <expressao> ";"

<cmd_return_void> ::= "return" ";"

- Em funções com tipo de retorno diferente de void, é obrigatório retornar um valor compatível com o tipo declarado.
- Em funções void, o comando return; é opcional e não contém expressão.

7.8. Estrutura geral de um programa MiniLua

Um programa MiniLua é composto por:

1. Zero ou mais declarações de função (incluindo funções auxiliares); e
2. Uma função especial main, do tipo void e sem parâmetros, que representa o ponto de entrada do programa.

Exemplo de programa completo:

```
function number fatorial(n : integer)
    if n <= 1 then
        return 1;
    else
        return n * fatorial(n - 1);
    end
```

```
end


function void main()
    local num : integer;
    print("Digite um numero:");
    num = inputNumber();
    print("Fatorial: ", fatorial(num));
end
```

A execução do programa inicia com a chamada implícita de main.