GRAMÁTICA DA LINGUAGEM

1.

Esta gramática define a linguagem **MiniLang**, uma linguagem baseada em estruturas comuns das linguagens C e Java. Ela é composta por declarações, comandos de controle de fluxo, entrada/saída, operadores, expressões e funções.

Observações sobre a estrutura da linguagem:

- Todos os comandos terminam com `;`
- Blocos de código são delimitados por `{` e `}`
- Expressões podem ser aninhadas com `( )`
- Identificadores (nomes de variáveis e funções) não podem começar com números
- A inclusão de bibliotecas é opcional, mas recomendada
  Exemplo: `#include <nome_da_biblioteca>`
- Todo programa deve conter obrigatoriamente a função `int main() { ... }`
  A execução sempre começa por ela.
  Importante: o `main` **não pode ser** `void`
- Funções e procedimentos auxiliares devem ser declaradas **após** o `main`
- Nenhum comando pode ficar fora de uma função — tudo precisa estar dentro de um bloco `{ }`

---

2. Tokens Reconhecidos

Palavras-chave:

```
int, string, boolean, const, void,
if, else, for, while, foreach, in,
```

```
break, continue, return,
getInt, getText, write,
true, false,
main, function,
#include <biblioteca>
```

Operadores:

- **Aritméticos:** `+`, `-`, `*`, `/`
- **Relacionais:** `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Lógicos:** `&&`, `||`, `!`
- **Incremento/Decremento:** `++`, `--`

Separadores:

```
;  ,  ( )  { }  [ ]
```

Literais:

```
Número: 0-9
String: "texto"
Booleano: true, false
```

---

3. Produções da Gramática

Início:

```
Start → (IncludeDecl)* MainMethod (FunctionDecl)* EOF
```

Inclusão de Bibliotecas:

```
IncludeDecl → #include <biblioteca>
```

Método Principal:

```
MainMethod → int main() Block
```

Bloco de Código:

```
Block → { (Statement)* }
```

Comandos (Statements):

```
Statement →
    VariableDecl |
    Assignment |
    Input |
    Output |
    IfStatement |
    WhileLoop |
    ForLoop |
    ForeachLoop |
    BreakStmt |
    ContinueStmt |
    FunctionDecl |
    FunctionCall |
    ReturnStmt
```

---

4. Declarações e Atribuições

Declaração de Variável:

```
VariableDecl → [const] (int | string | boolean) ID [= Expression] ;
```

Atribuição:

```
Assignment → ID = Expression ;
           | ID++ ;
           | ID-- ;
```

---

5. Expressões

Hierarquia de Expressões:

```
Expression     → AddExpr
AddExpr        → MultExpr ((+ | -) MultExpr)*
MultExpr       → LogicExpr ((* | /) LogicExpr)*
LogicExpr      → LogicTerm (|| LogicTerm)*
LogicTerm      → LogicFactor (&& LogicFactor)*
```

```
LogicFactor    → [!] Term ((== | != | < | > | <= | >=) Term)?
```

Termo:

```
Term →
    NUMBER |
    STRING_LITERAL |
    TRUE |
    FALSE |
    ID |
    ( Expression ) |
    FunctionExpr
```

Chamada de função como expressão:

```
FunctionExpr → ID ( [ArgList] )
```

Lista de Argumentos:

```
ArgList → (ID | NUMBER | STRING_LITERAL | TRUE | FALSE) (, ...)
```

---

6. Entrada e Saída

Entrada:

```
Input → getInt(ID) ; | getText(ID) ;
```

Saída:

```
Output → write( OutputExpr ) ;
OutputExpr → OutputTerm (+ OutputTerm)*
OutputTerm → STRING_LITERAL | ID | NUMBER
```

---

7. Comandos de Controle de Fluxo

If / Else If / Else:

```
IfStatement →
    if ( Expression ) Block
    [ else IfStatement | else Block ]
```

Laço While:

```
WhileLoop → while ( Expression ) Block
```

Laço For:

```
ForLoop →
    for ( VariableDecl | Assignment ; Expression ;
SimpleAssignmentExpr ) Block
```

Laço Foreach:

```
ForeachLoop →
    foreach ( int | string | boolean ID in ID ) Block
```

Break e Continue:

```
BreakStmt → break ;
ContinueStmt → continue ;
```

---

8. Funções e Procedimentos

Declaração de Função:

```
FunctionDecl →
    (void | int | string | boolean) ID ( [ParamList] ) Block
```

Parâmetros:

```
ParamList → Param (, Param)*
Param     → (int | string | boolean) ID
```

Chamada de Função:

```
FunctionCall → ID ( [ArgList] ) ;
```

Retorno:

```
ReturnStmt → return Expression ;
```
