A EBNF definida é bastante abrangente e se alinha bem com o objetivo da linguagem musical. Agora, vamos trabalhar na **documentação detalhada para o README do repositório** com base nessa EBNF.

README - Linguagem Musical

Nome da Linguagem

Melodify - Uma linguagem de programação projetada para compor música de forma simples e intuitiva.

Motivação

A música é uma linguagem universal. No entanto, a composição musical pode ser complexa para quem não é músico ou não possui ferramentas avançadas. **Melodify** foi criada para simplificar a criação de músicas, oferecendo uma sintaxe acessível e intuitiva, com comandos básicos que permitem gerar notas, acordes e até padrões rítmicos.

Características Principais

1. **Declaração de Variáveis**: Manipule variáveis para controlar o andamento, volume e outras propriedades musicais.
2. **Condicionais e Loops**: Use estruturas de controle para criar padrões musicais dinâmicos.
3. **Instruções Musicais**: Toque notas, acorde, defina instrumentos e ajuste o tempo de forma simples.
4. **Funções Personalizadas**: Crie blocos reutilizáveis de código para escalar suas composições.
5. **Exportação para MIDI**: Compile o código para um arquivo `.midi`, que pode ser reproduzido em qualquer software de música.

Gramática Formal (EBNF)

```
program = statement_list ;

statement_list = { statement } ;

statement =
      variable_declaration
    | assignment
    | musical_instruction
    | conditional
    | loop
    | function_declaration
    | function_call
    ;

variable_declaration =
      'set' IDENTIFIER [ '=' ] ( expression | STRING ) ';' ;
```

```
assignment =
      IDENTIFIER '=' expression ';' ;

musical_instruction =
      play_note
    | play_chord
    | set_musical_property ;

play_note = 'play' NOTE DURATION ';' ;

play_chord = 'play' 'chord' CHORD_NAME DURATION ';' ;

set_musical_property =
      'set' musical_property ';' ;

musical_property =
      'tempo' expression
    | 'instrument' STRING
    | 'volume' expression ;

conditional =
      'if' '(' condition ')' '{' statement_list '}' [ 'else' '{' statement_list
'}' ] ;

loop =
      'repeat' NUM '{' statement_list '}' ;

function_declaration =
      'function' IDENTIFIER '(' ')' '{' statement_list '}' ;

function_call =
      IDENTIFIER '(' ')' ';' ;

condition =
      expression relational_operator expression
    | condition logical_operator condition
    ;

relational_operator = '<' | '>' | '==' | '!=' ;

logical_operator = '&&' | '||' ;

expression =
      expression '+' term
    | expression '-' term
    | term
    ;

term =
      term '*' factor
    | term '/' factor
    | factor
    ;
```

```
factor =
    | '-' factor
    | NUM
    | IDENTIFIER
    | 'tempo'
    | 'instrument'
    | 'volume'
    | NOTE
    ;

IDENTIFIER = /[a-zA-Z_][a-zA-Z0-9_]*/ ;
NUM = /[0-9]+/ ;
STRING = /"[^"]*"/ ;

NOTE = /C[0-9]|C#[0-9]|D[0-9]|D#[0-9]|E[0-9]|F[0-9]|F#[0-9]|G[0-9]|G#[0-9]|A[0-9]|A#[0-9]|B[0-9]/ ;

CHORD_NAME = /[A-G](#)?_(major|minor|diminished)/ ;
DURATION = 'whole' | 'half' | 'quarter' | 'eighth' ;
```

Exemplos de Código

1. Definindo propriedades musicais

```
set tempo 120;
set instrument "piano";
set volume 80;
```

2. Intruções Musicais

As notas devem ser definidas com a oitava `(C4, D4, E4, F4, G4, A4, B4)` e a duração `(whole, half, quarter, eighth)`.

Antes de tocar uma nota, é necessário usar o comando `play`.

Os acordes são definidos com o nome do acorde `(C_major, D_minor, E_minor, F_major, G_major, A_minor, B_diminished)`.

Antes de tocar um acorde, é necessário usar o comando `play chord`.

```
play C4 quarter;
play chord G_major half;
play C4 quarter;
```

3. Declaração de Variável

```
set myNumber = 42;
```

4. Condicionais

```
set tempo 120;

if (tempo > 100) {
    play C4 quarter;
} else {
    play G4 quarter;
}
```

5. Loops

```
repeat 4 {
    play C4 quarter;
}
```

6. Loops Aninhados

```
repeat 2 {
    play C4 quarter;

    repeat 3 {
        play E4 eighth;
        play G4 eighth;
    }

    play F4 half;
}
```

7. Declaração e Chamada de Funções

```
set tempo 120;
set instrument "guitar";

function playChord() {
    play chord C_major quarter;
    play chord E_minor quarter;
```

```
    play chord G_major half;
}

playChord();
playChord();
```

8. Condições com Operadores Lógicos

```
set tempo 120;
set volume 80;

if (tempo > 100 && volume < 90) {
    play A4 half;
    play C4 half;
} else {
    play B4 half;
}
```

```
set tempo 120;
set volume 80;

if (tempo > 100 || volume < 90) {
    play A4 half;
    play C4 half;
} else {
    play B4 half;
}
```

9. Operações Aritméticas

Adição

```
set tempo 60 + 60;
set instrument "flute";

if (tempo > 100) {
    play C4 quarter;
    play E4 quarter;
} else {
    play G4 quarter;
}
```

Subtração

```
set tempo 120-10;
set instrument "flute";

if (tempo > 100) {
    play C4 quarter;
    play E4 quarter;
} else {
    play G4 quarter;
}
```

Multiplicação

```
set tempo 60*2;
set instrument "flute";

if (tempo > 100) {
    play C4 quarter;
    play E4 quarter;
} else {
    play G4 quarter;
}
```

Divisão

```
set tempo 220/2;
set instrument "flute";

if (tempo > 100) {
    play C4 quarter;
    play E4 quarter;
} else {
    play G4 quarter;
}
```

Passo a Passo do Projeto

1. **Tratamento Léxico e Sintático**:
    - O tratamento léxico e sintático é realizado pelos arquivos `lexer.l` e `parser.y`, que utilizam Flex e Bison, respectivamente.
    - Esses arquivos, junto com outros componentes do projeto, são compilados com o `Makefile`:
2. **Como Compilar o Projeto**:
    - Para compilar o executável melodify, basta rodar:

```
make
```

- Quando necessário, limpe os arquivos gerados com:

```
make clean
```

3. **Executar um Código da Linguagem**:
   - Escreva um arquivo contendo código na linguagem `(exemplo: program.melody)` e execute:

     ```
     ./melodify program.melody
     ```

     - O comando acima irá gerar uma `AST` (Abstract Syntax Tree) no arquivo `ast.yaml`.

4. **Tratamento Semântico e Geração de Código**:
   - Após a `AST` ser gerada, o arquivo `main.py` irá processá-la utilizando os arquivos `parser.py` e `nodes.py`.
   - Esse processamento realiza o **tratamento semântico** e chama as funções `generate_code` dos nós da `AST` para criar o código de JVM responsável por gerar o arquivo MIDI, `output.mid`.

5. **Testar o Arquivo MIDI Gerado**:
   - Para tocar o arquivo MIDI gerado (`output.mid`), utilize o script `play_midi.py`:

     ```
     python3 play_midi.py
     ```

     - O script inicializa o mixer do Pygame, carrega o arquivo MIDI e o executa.

6 **Resumo do Fluxo**:

```
- Escreva o código na linguagem em um arquivo .melody.

- Compile e execute o código com melodify para gerar a AST (ast.yaml).

- Rode o main.py para processar a AST, gerar o código em Java e produzir o
arquivo MIDI (output.mid).

- Teste o arquivo MIDI com o script play_midi.py.
```

Essa sequência garante a geração correta do arquivo MIDI e sua execução.

Conjunto de Testes

Incluímos um conjunto de testes na pasta `tests/` com exemplos como:

- Composição básica (`basic.melody`).
- Uso de loops (`loop.melody`).
- Funções avançadas (`functions.melody`).

Execute os testes com:

```
./melodify tests/basic.melody
```
