

Sumário

<b>1. REQUISITOS E EXECUÇÃO DO PROJETO .....</b>	<b>2</b>
<b>2. SOBRE A LINGUAGEM CRIADA.....</b>	<b>3</b>
<b>3. ESTRUTURA BÁSICA DO PROGRAMA .....</b>	<b>7</b>
<b>    3.1 Comentários .....</b>	<b>8</b>
<b>    3.2 Declarações de variáveis .....</b>	<b>9</b>
<b>    3.2 Atribuições de variáveis.....</b>	<b>10</b>
<b>    3.3 Impressão de valores com cout .....</b>	<b>11</b>
<b>    3.4 Leitura de valores com cin.....</b>	<b>12</b>
<b>    3.5 Condições if e else .....</b>	<b>13</b>
<b>    3.6 Repetição while.....</b>	<b>14</b>
<b>    3.7 Repetição for.....</b>	<b>15</b>
<b>    3.8 Modo de depurar com debug.....</b>	<b>16</b>

## 1. REQUISITOS E EXECUÇÃO DO PROJETO

O compilador foi feito utilizando ANTLR com Java e a IDE IntelliJ.

Para abrir o código fonte e executar direto de uma IDE, baixe em:  
[Compilador-com-ANTLR-e-Java - Github](#)

É recomendando possuir a versão 17.0.11 da JDK (Java Development Kit)

Sem ser necessário a utilização de uma IDE, é possível baixar a aplicação executável do projeto em: [Compilador-com-ANTLR-e-Java - Release Github](#)

Após descompactar a pasta baixada, terá um arquivo chamado **Compilador-com-ANTLR-e-Java.jar** e uma pasta chamada **testes**. Os testes é onde estarão os arquivos para executar no compilador, que devem ter a extensão **.medjed** (porém podem ser editados em qualquer editor de texto). Na pasta descompactada abra o terminal e execute o projeto com:

```
java -jar Compilador-com-ANTLR-e-Java.jar testes/nome-do- arquivo-que-  
ira-utilizar.medjed
```

*OBS: Lembre-se de possuir o Java instalado no seu computador.*

Resultado:

```
PS C:\Users\Thiago Ianuch\Downloads\Projetos\Aplicação executável> java -jar Compilador-com-ANTLR-e-Java.jar testes/Entrada.medjed  
Usando arquivo: testes/Entrada.medjed  
Nenhum erro léxico encontrado!  
Nenhum erro sintático encontrado!  
Nenhum erro semântico encontrado!  
  
50 - 20 = 30  
O valor de: X + Y é: 7517  
O valor de Y + Z é: 100  
  
Esse é um exemplo de STRING  
E esse um exemplo de booleano: true  
  
O valor de X agora é '10', que é menor que 50  
Digite um valor inteiro: 40  
  
50 + 90 * 40 / 2 = 1850  
(50 + 90) * 40 / 2 = 2800  
(50 + (90 * 40)) / 2 = 1825
```

## 2. SOBRE A LINGUAGEM CRIADA

A linguagem criada segue uma sintaxe baseada no C++ e é case-sensitive.

Contendo:

- Comentários
- Declarações de variáveis
- Atribuições de variáveis
- Comando de impressão e leitura
- Condições
- Repetições

Os arquivos para serem executados devem possuir a extensão “.medjed”. Apesar da extensão diferente, podem ser facilmente abertos e editados em qualquer editor de texto.

A análise léxica e sintática é feita pelo ANTLR, porém o projeto possui um arquivo chamado Lexico.java e Sintatico.java que reescreve o código do ANTLR, para permitir que apareça uma mensagem de erro léxico ou sintático em português (somente a mensagem de que ocorreu um erro irá aparecer em português, o erro em si não está traduzido).

Além disso contem um arquivo Semantico.java, onde é feito toda a análise semântica.

Além disso foi também tentado criar um interpretador após a análise semântica, encontrado no arquivo Interpretador.java, que executa os comandos necessários. Porém o Interpretador ainda não executa as repetições **WHILE** e **FOR**, isso ocorre pois não foi possível armazenar quais comandos deveriam ser repetidos corretamente. Porém mesmo que esses dois comandos não sejam interpretados, a análise semântica ainda ocorre. Além disso o interpretador pode não executar corretamente os **IF** e **ELSE** se forem executados com valores de entrada pelo usuário, por tanto é recomendado evitar IF e ELSE com entradas do usuário.

Se caso possuir algum erro, o compilador irá dizer quais erros ocorreram, caso contrário será exibido que não ocorreu nenhum erro e os comandos serão executados corretamente.

Exemplo de código compilado na linguagem com erros léxicos, sintáticos e semânticos:

```
BeginPlay()
{
    @
    int X = 50.1;

    cout << Y;

    cout << "Teste"
}
```

```
Usando arquivo padrão: testes/Entrada.medjed

Foram encontrados erros léxicos na execução do programa:
Erro na linha 3, coluna 4: token recognition error at: '@'

Foram encontrados erros sintáticos na execução do programa:
Erro na linha 9, coluna 0: extraneous input '}' expecting ';' , '<<'

Foram encontrados erros semânticos na execução do programa:
Erro na linha 4, coluna 8: a variável 'X' foi declarada com o valor incorreto!
Erro na linha 6, coluna 12: a variável 'Y' não foi declarada!

Process finished with exit code 0
```

Exemplo de código compilado na linguagem que não possui nenhum tipo de erro:

```
BeginPlay()
{
    int X, VariavelInt, A;
    int Y = 50;
    float Z = 50.0, VariavelFloat;
    string MinhaString = "Esse é um exemplo de STRING";
    bool VariavelBool = true;

    VariavelInt = 50, A = 20;
    cout << VariavelInt << " - " << A << " = " << VariavelInt - A;

    cout << "\n\nO valor de: X + Y é: " << X + Y << "\n"; // Isso irá gerar um valor inesperado, pois a variável X não foi inicializada
    cout << "O valor de Y + Z é: " << Y + Z << "\n"; // Isso irá mostrar o valor '100' e não '100.0', pois como as casas decimais do resultado são '0' ele irá converter para INTEIRO
    cout << "\n" << MinhaString;
    cout << "\nÉ esse um exemplo de booleano: " << VariavelBool;

    X = 10;
    if(X > 50)
    {
        cout << "\n\nO valor de X agora é '" << X << "', que é maior que 50";
    }
    else
    {
        cout << "\n\nO valor de X agora é '" << X << "', que é menor que 50";
    }

    cout << "\n\nDigite um valor inteiro: ";
    cin >> X;

    cout << "\n50 + 90 * 40 / 2 = " << 50 + 90 * 40 / 2; // O resultado deve ser 1850
    cout << "\n(50 + 90) * 40 / 2 = " << (50 + 90) * 40 / 2; // O resultado deve ser 2800
    cout << "\n(50 + (90 * 40)) / 2 = " << (50 + (90 * 40)) / 2; // O resultado deve ser 1825
}
```

### Usando arquivo padrão: testes/Entrada.medjed

```
Nenhum erro léxico encontrado!
Nenhum erro sintático encontrado!
Nenhum erro semântico encontrado!
```

```
50 - 20 = 30
```

```
O valor de: X + Y é: 16494
O valor de Y + Z é: 100
```

```
Esse é um exemplo de STRING
E esse é um exemplo de booleano: true
```

```
O valor de X agora é '10', que é menor que 50
```

```
Digite um valor inteiro: 60
```

```
50 + 90 * 40 / 2 = 1850
(50 + 90) * 40 / 2 = 2800
(50 + (90 * 40)) / 2 = 1825
```

Devido a sintaxe e aos comportamentos serem semelhantes ao do C++, o código dentro de “BeginPlay()” pode ser facilmente adaptado e executado em um compilador de C++ e exibir um resultado semelhante ao de cima.

```
main.cpp
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int X, VariavelInt, A;
8     int Y = 50;
9     float Z = 50.0, VariavelFloat;
10    string MinhaString = "Esse é um exemplo de STRING";
11    bool VariavelBool = true;
12
13    VariavelInt = 50, A = 20;
14    cout << VariavelInt << " - " << A << " = " << VariavelInt - A;
15
16    cout << "\n\nO valor de: X + Y é: " << X + Y << "\n"; // Isso irá gerar um valor inesperado, pois a variável X não foi inicializada com 10
17    cout << "O valor de Y + Z é: " << Y + Z << "\n"; // Isso irá mostrar o valor '100' e não '100.0', pois a variável Z é float
18    cout << "\n" << MinhaString;
19    cout << "\nÉ esse um exemplo de booleano: " << VariavelBool;
20
21    X = 10;
22    if(X > 50)
23    {
24        cout << "\n\nO valor de X agora é '" << X << "'", que é maior que 50";
25    }
26    else
27    {
28        cout << "\n\nO valor de X agora é '" << X << "'", que é menor que 50";
29    }
30
31    cout << "\n\nDigite um valor inteiro: ";
32    cin >> X;
33
34    cout << "\n\n50 + 90 * 40 / 2 = " << 50 + 90 * 40 / 2; // O resultado deve ser 1850
35    cout << "\n(50 + 90) * 40 / 2 = " << (50 + 90) * 40 / 2; // O resultado deve ser 2800
36    cout << "\n(50 + (90 * 40)) / 2 = " << (50 + (90 * 40)) / 2; // O resultado deve ser 1825
37
38 }
```

```
50 - 20 = 30
O valor de: X + Y é: 604921412
O valor de Y + Z é: 100
Esse é um exemplo de STRING
E esse é um exemplo de booleano: 1
O valor de X agora é '10', que é menor que 50
Digite um valor inteiro: 60
50 + 90 * 40 / 2 = 1850
(50 + 90) * 40 / 2 = 2800
(50 + (90 * 40)) / 2 = 1825
...Program finished with exit code 0
Press ENTER to exit console.
```

Compilado com: [https://www.onlinegdb.com/online\\_c++\\_compiler](https://www.onlinegdb.com/online_c++_compiler)

### 3. ESTRUTURA BÁSICA DO PROGRAMA

O programa deve obrigatoriamente começar com “BeginPlay()”, seguido da abertura de chaves que deve ser fechada no final do programa. Dentro das chaves é possível declarar variáveis, atribuições, condições, repetições e comandos de impressão e leitura.

Exemplo:

```
BeginPlay()
{
    // Declarações devem ser incluídas aqui
}
```

### 3.1 Comentários

Os comentários podem ser feitos utilizando “//”. Não é possível fazer comentários de múltiplas linhas, somente de linha única.

Exemplo:

```
// Exemplo de comentário  
// Esse é outro comentário
```

### 3.2 Declarações de variáveis

As variáveis podem ser declaradas em qualquer parte do código, porém elas possuem escopo, por tanto não é possível declarar a mesma variável dentro do mesmo escopo.

A declaração pode ser feita sem inicializar o valor (mesmo que não seja inicializado, a variável ainda terá um valor padrão) ou declaração com inicialização direta. Além disso é permitido declarar mais de uma variável por linha, desde que elas sejam do mesmo tipo.

Exemplo:

```
tipo nome_da_variavel;
tipo nome_da_variavel1, nome_da_variavel2, nome_da_variavel3;

tipo nome_da_variavel = valor;
tipo nome_da_variavel1 = 50, nome_da_variavel2 = 40;
```

O nome da variável deve começar obrigatoriamente com uma letra (maiúscula ou minúscula), podendo então ser seguido de qualquer letra ou número. Os tipos e valores aceitos são:

- **INT** - números inteiros positivos ou negativos ('-?[0-9]+).
- **FLOAT** - números decimais positivos ou negativos, deve ser utilizado ponto em vez de vírgula ('-?[0-9]+.[0-9]+).
- **BOOL** – valor booleano verdadeiro ou falso ('true' | 'false'). Diferente de C++ e outras linguagens, o valor 0 e 1 não é aceito.
- **STRING** – qualquer tipo de texto, desde que esteja entre aspas. A quebra de linha “\n” também é aceita ("\" (\"\\r|\"\\n)\* '\"").

Exemplo:

```
int X, Y = 50, Z = 20;

float A, B, C = 50.1 + 10;

string Texto = "Texto";

bool booleano = true, booleano2 = false;
```

### 3.2 Atribuições de variáveis

As variáveis também podem ter seu valor atribuído após a declaração, para isso é necessário informar o nome da variável, seguido do operador de atribuição “=” e do valor. É possível fazer varias atribuições na mesma linha e as variáveis que estão tendo o valor atribuído não precisam ser do mesmo tipo.

Exemplo:

```
int X, Y;  
float Z;  
  
X = 50, Y = 20, Z = 50.1;
```

### 3.3 Impressão de valores com cout

Para imprimir se utiliza “cout” no início, seguido do operador “<<” e do valor que irá ser impresso. É possível imprimir mais de um valor por linha e o ponto e vírgula é necessário no final.

Exemplo:

```
cout << "Exemplo do uso de cout\n";  
  
cout << "É possível imprimir textos, números e expressões aritméticas:  
" << 50 + 40 << "\n";  
  
cout << "A impressão de variáveis também é aceita: " << X << " + " <<  
Y << " = " << X + Y << "\n";
```

**OBS:** se tentar realizar a impressão de uma variável/valor INT somado com uma variável/valor FLOAT, caso as casas decimais sejam 0, o programa irá exibir o número como sendo inteiro, caso contrário irá exibir como um número decimal, assim exibindo as casas decimais. Isso é semelhante a C++.

```
A soma de 50 + 50.0 irá exibir: 100  
A soma de 50 + 50.1 irá exibir: 100.1
```

### 3.4 Leitura de valores com cin

A leitura de valores é feita utilizando “cin” no início, seguido do operador “>>” e da variável que irá receber o valor. É possível passar mais de um variável que irá receber determinado valor e o usuário terá que digitar todos eles.

Exemplo:

```
cout << "\n\nDigite um valor inteiro para X e depois para Y: ";
cin >> X >> Y;
```

Caso o valor que seja digitado não seja compatível com o valor da variável, o programa será encerrado automaticamente exibindo erro sem executar os comandos seguintes, diferente de C++ que aceita a leitura de um valor mesmo que seja o tipo incorreto da variável, fazendo com que o valor seja convertido para algum valor padrão.

```
Digite um valor inteiro: aaa
O programa foi encerrado pois o valor digitado é inválido para a variável 'X' de tipo (INT)!
```

### 3.5 Condições if e else

As condições podem ser feitas utilizando **IF**, seguido da abertura de parênteses, que deve conter dentro uma expressão booleana (**valor/variável, operador de comparação, valor/variável**), seguindo então de abre chaves, podendo conter declarações, atribuições e outras coisas dentro, sendo necessário fechar as chaves no final do bloco.

O **ELSE** não é obrigatório, mas também pode ser utilizado e será executado caso a condição do IF seja falsa. **ELSE IF (condição se não se)** não é aceito pelo programa.

Os operadores de comparação aceitos são:

- == (igual)
- != (diferente)
- > (maior)
- >= (maior ou igual)
- < (menor)
- <= (menor ou igual)

O tipo **INT** e **FLOAT** podem utilizar qualquer um dos operadores, enquanto **BOOL** e **STRING** só podem utilizar o operador ‘==’ ou ‘!=’. Não é possível ter mais de uma condição utilizando operadores **&&** ou **||**.

Exemplo:

```
int X = 5;

if(10 / 2 * 3 > X)
{
    cout << "É verdadeiro\n";
}
else
{
    cout << "É falso\n";
}

if("Texto" == "Texto")
{
    cout << "Texto é igual a Texto";
}
```

### 3.6 Repetição while

Uma das formas de repetição é utilizando “while”, que deve conter entre parênteses a condição e dentro das chaves as declarações que serão executadas na repetição “while”.

Assim como em outras linguagens, a condição de parada da repetição “while” não é obrigatória, porém isso significa que será executado infinitamente.

A repetição é analisada semanticamente, porém o interpretador ainda não realiza os comandos todas as vezes, por tanto ao executar os comandos serão executados somente uma vez.

Exemplo:

```
int X = 1;

while(X <= 10)
{
    cout << X + 1 << "\n";

    X++ // PODE SER TAMBÉM X = X + 1;
}
```

### 3.7 Repetição for

A repetição “for” também é semelhante ao C++ e é separada em “3 blocos ” diferentes.

O primeiro bloco permite a declaração de qualquer variável, desde que sejam do mesmo tipo, no segundo bloco é necessária alguma condição, e no terceiro e último bloco é declarado as expressões unárias de incremento (++) ou decremento (--).

```
for(bloco1; bloco2; bloco3)
{
    // Bloco do for
}
```

Por tanto, um “for” válido é declarado da seguinte forma:

```
for(int i = 0; i < 20; i++)
{
    cout << i << "\n";
}
```

O “for” abaixo também é válido na linguagem e também é considerado válido em C++, mesmo que a repetição seja infinita. Isso ocorre pois o primeiro bloco permite a declaração de qualquer tipo de variável, mesmo que seja do tipo STRING, no segundo bloco é necessária uma condição e no terceiro bloco é necessário as expressões unárias, que neste caso funcionam, pois, as variáveis ‘j’ e ‘k’ já foram declaradas no escopo global.

```
int j = 0, k = 0;
for(string teste = "teste1", teste2="teste2"; teste == "100"; j++, k--)
{
    cout << i << "\n";
}
```

### 3.8 Modo de depurar com debug

Ao utilizar “--debug” logo no inicio do código, o compilador entra em um modo para “depurar”, exibindo todas as variáveis que foram declaradas e comandos que serão executados. Só é possível utilizar “--debug” no inicio do código, antes de BeginPlay() e ele não precisa de ponto e vírgula no final.

Exemplo:

```
--debug

BeginPlay()
{
    // Resto do código
}
```

```
Variáveis declaradas por escopo:
Escopo nível 0:
    INT A = 30
    INT B = 50
    INT C = 20
    FLOAT D = 20964.625
    FLOAT E = 16605.396
    FLOAT F = 40.0
    STRING variavelTexto = "Alteração no texto"
    STRING texto = "Outro exemplo de string"
    BOOL booleanoVerdadeiro = true
    BOOL booleanoFalse = false
Escopo nível 1:
    INT XYZ = 500

Lista de comandos:
Tipo: AbrirEscopo, Valor: null
Tipo: Declarar, Valor: [59 47 41]
Tipo: Declarar, Valor: [61 47 41]
Tipo: Declarar, Valor: [61 47 41]
Tipo: Declarar, Valor: [59 47 41]
Tipo: Declarar, Valor: [61 47 41]
Tipo: Declarar, Valor: [61 47 41]
Tipo: Declarar, Valor: [59 47 41]
Tipo: Declarar, Valor: [61 47 41]
Tipo: Declarar, Valor: [61 47 41]
Tipo: Declarar, Valor: [69 48 41]
Tipo: Atribuir, Valor: [71 48 41]
Tipo: Atribuir, Valor: [71 48 41]
Tipo: Atribuir, Valor: [69 48 41]
Tipo: FecharEscopo, Valor: null
```