Especificação da Linguagem

BRLanguage

Programming Language Project for Compiladores 2020.1 - UFAL - A BRLanguage foi pensanda para ser uma linguagem simples, fácil de implementar algorítmos mais básico e para aprender conceitualmente sobre programação, além de ser toda em português o que permite um aprendizado de conceitos como tipos de dados e estruturas de controles de forma mais simples.

1. Características Gerais:

- A linguagem é toda em português;
- A linguagem pertece ao paradigma Imperativo;
- O método utilizado de implementação é a compilação.

2. Estrutura Geral de Um Programa

- Para se criar um programa em BRLanguage deve-se escrever um código, obdecendo as regras da linguagem, e colocar a extensão ".brl";
- BRLanguage funciona com um conjunto de funções, onde a função que começa o programa é denominada "Principal". A função Principal retorna um tipo definido pelo programador e possui a sintaxe abaixo:

```
VAZIO FUNCAO PRINCIPAL(){
  RETORNE ;
};
```

- Ao final de cada comando deve-se colocar o delimitador de instrução ";" ;
- Os delimitadores de escopo são:
  1. Inicar: "{";
  2. Finalizar: "}";
- Um programa em .brl deve seguir as regras de identação para um melhor entendimento;
- A linguagem permite comentários por linha utilizando o delimitador de comentários: "#".

3. Características Léxicas:

- Identificadores não podem começar com numeros e recomendasse que sejam escritos em minúsculo;
- Identificadores podem conter "-" ou "_" na composição mas nenhum outro símbolo;
- Identificadores devem ter no máximo tamanho 32 caracteres e não podem conter caracteres especiais, apenas o alfabeto latino padrão, e não devem coincidir com as palavras reservadas da linguagem.

- Lista de palavras reservadas da linguagem:

Especificação de Tipo	Função	Comandos	Operadores Lógicos
INTEIRO	FUNCAO	SE	VERDADE
FLUTUANTE	PRINCIPAL	OU_SE	FALSO
CARACTERE	LER	SENAO	NAO
BOOLEANO	IMPRIMIR	FAZ	E
CARACTERES	RETORNE	ENQUANTO	OU
VAZIO		ITERADOR	

4. Especificação de Tipos:

- A linguagem é tipada estaticamente, o que significa que toda variavél precisa indicar o seu tipo ao ser declarada e esse tipo fica inculado a varíavel durante toda a sua vida;
- Lista de tipos definidos em BRLanguage:
  - INTEIRO - Inteiro de x tamanho em bits
  - FLUTUANTE - Ponto flutuante de x tamanho em bits
  - CARACTERE - Caractere
  - BOOLEANO - Booleano
  - CARACTERES - cadeia de caractere de x tamanho máximo de caracteres
  - *tipo_do vetor identificador_do_vetor*[tamanho] - Arranjos unidimencionais de qualquer tipo
  - VAZIO - Tipo de dado que representa função sem retorno.

1. Operação Suportadas:

   - Todos os tipos primitivos suportam as operações descritas na tabela abaixo, com os valores de seu tipo específico, ou segundo as regras de

coerções:

Tipo	Operações Suportadas
INTEIRO	atribuição, aritméticas, relacionais
FLUTUANTE	atribuição, aritméticas, relacionais
CARACTERE	atribuição, relacionais
BOOLEANO	atribuição, relacionais
CARACTERES	atribuição, relacionais, concatenação
vetor	atribuição

2. Coerção:
    - A linguagem prevê coerção autômatica entre o tipo INTEIRO e FLUTUANTE. Onde, se um número do tipo ponto flutuante for atribuído a uma variável do tipo inteiro, o valor da variável será apenas a parte inteira do ponto flutuando e na forma de um inteiro se atribuído a um ponto flutuante, a parte inteira será o número e a parte decimal será zero.

5. Variáveis:

1. Declaração:
    1. A declaração da variável tem a seguinte regra geral: ``` TIPO identificador_da_variável;
       ```
2. Iniciação Padrão:
    1. Pode ser feita junto da declaração da seguinte forma: ```
       TIPO identificador_da_variável = valor_do_tipo;
       ```
        1. Enquanto não tiver um valor atribuído a uma variável, ela terá um valor inicial padrão como se segue:
        - INTEIRO: iniciado com valor zero(0);
        - FLUTUANTE: iniciado com valor zero(0.0);
        - CARACTERE: iniciado com valor " " (espaço em branco, 32 no código ASCII);
        - BOOLEANO: iniciado com valor FALSE;
        - CARACTERES: iniciado com o valor "" (cadeia de caracteres vazia);
        - *VETOR*: iniciado com o valor padrão do tipo no qual foi definido.
            1. Escopo:
            2. Na BRlanguege, todas as variáveis são globais, logo vistas em todo o programa.

6. Operadores:

- Temos os seguintes operadores na linguagem:
  Aritméticos:

Operador	Operação
+	adição
-	subtração
*	multiplicação
\/	divisão
-	negativo unário

Relacionais:

Operador	Operação
>	maior que
\<	menor que
\==	igual
!=	diferente de
>=	maior ou igual

Operador	Operação
\<=	memor ou igual

Lógicos:

Operador	Operação
VERDADE	verdadeiro
FALSO	falso
NAO	negação
E	conjunção
OU	disjunção

Concatenção de Cadeias de Caracteres:

Operador	Operação
&	concatenação

1. Precedência e Associatividade:

- Conforme a tabela abaixo, da ordem da mais alta para a mais baixa vemos a Precedência e os operadores segue a Associatividade espeficificada:

Operador	Associatividade
NAO	direita para esquerda
- (negativo unário)	direita para esquerda
* \/	esquerda para direita
+ -	esquerda para direita
\< \<= > >=	esquerda para direita
\== !=	esquerda para direita
E	esquerda para direita
OU	esquerda para direita

- Podemos utilizar parênteses para alterar a precedência dos operadores acima.

7. Instruções:

1. Estrutura condicional de uma e duas vias:

Estrutura	Comando em Brl
if	SE
if \/ else	SE \/ SENAO
if \/ elseif \/ else	SE \/ MAS_SE \/ SENAO

   - A instrução SE controla a fluxo condicional. O bloco da instrução SE é executado se o valor da expressão for diferente de zero. Temos duas sintaxes para essa estrutura:

```
SE(expressao_logica){
 lista_de_comandos;
};
```

ou

```
SE(expressao_logica){
 lista_de_comandos_1;
}
SENAO(espressao_logica){
 lista_de_comandos_2;
};
```

- Usamos uma expressao lógica para controlar essa estrutura condicional.
- Se uma forma mais abrangente de verificação for necessária temos o SE's aninhados, utilizando o comando SENAO.

```
SE(expressao_logica){
 lista_de_comandos_1;
}
MAS_SE(expressao_logica){
 lista_de_comandos_2;
}
SENAO(espressao_logica){
 lista_de_comandos_3;
};
```

2. Estrutura iterativa com controle lógico:

Estrutura	Comando em Brl
while	ENQUANTO
do while	FAZER ENQUANTO

- A estrutura ENQUANTO permite que se repita uma instrução até que seja verificado que uma expressão é falsa. Sintaxe:

```
ENQUANTO(expressao_logica){
 lista_de_comandos;
};
```

- A estrutura FAZ ENQUANTO permite que se repita uma instrução pelo menos uma vez, mesmo que a expressão seja falsa.

  Sintaxe:

```
FAZER{
 lista_de_comandos;
}ENQUANTO(expressao_logica);
```

3. Estrutura iterativa controlada por contador:

Estrutura	Comando em Brl
for	ITERADOR

- A estrutura ITERADOR permite que seja repetida uma instrução composta por um número específico de vezes. O corpo da estrutura é executado zero ou mais vezes até que a condição seja verificada como falsa.

  Sintaxe:

```
ITERADOR(inicialização, condicao, incremento){
 lista_de_comandos;
};
```

4. Entrada e Saida

Entrada	Saida

Entrada	Saida
LER	IMPRIMIR

- IMPRIMIR é a instrução de escrita na tela, recebe como parâmetro um dos tipos da linguagem , permite a formatação atravez da concatenação de cadeia de caracteres e variáveis.

  Sintaxe:

  ```
  IMPRIMIR(frase_para_imprimir_na_tela);
  ```

- LER é a instrução que lê algo, deve receber o tipo a ser lido e sua variável para referência.

  ```
  LER(TIPO, variavel_para_referencia);
  ```

8. Atribuição:

A atribuição é feita com operador "=", com a associatividade da direita para a esquerda, sempre.

Exemplo: ```

a = b;

``` *o valor da variável b está agora sendo atribuído para a*

## 9. Funções:

- Na BRLanguage, temos a função PRINCIPAL e ela é obrigatória pois é por ela que o compilador começa sua execução.
- As funções devem ser definidas antes de serem chamadas.
- A passagem de parâmetro ocorre por valor na liguagem.
- Para Definir uma função utilizamos a palavra reservada FUNCAO antes do seu identificador e depois do seu tipo de retorno.
- Para o retorno de funções, utilizamos a palavra reservada RETORNE na última linha do escopo da função

- A definição segue a regra abaixo:

```
TIPO_DE_RETORNO FUNCAO nome_da_função(parametros){
  comandos;
  ...
  ...
  ...
  RETORNE TIPO_DE_RETORNO

}
```

## 10. Exemplos:

- Alô Mundo:

```
VAZIO FUNCAO PRINCIPAL(){
  IMPRIMIR("Alô Mundo!");
  RETORNE ;
}
```

- Fibonacci:
```

```
VAZIO FUNCAO fibonacci(INTEIRO limite){
  INTEIRO contador;
  INTEIRO um;
  INTEIRO dois;
  INTEIRO tres;

  um = 1;
  dois = 1;

  SE(limite == 0){
     IMPRIMIR("0");
  }

  ENQUANTO(contador < limite){
     SE(contador < 2){
        IMPRIMIR("1");
     }

     SENAO{
        tres = um + dois;
        um = dois;
        dois = tres;
        IMPRIMIR(tres & " ");
     }
  }

  contador = contador + 1;
}

VAZIO FUNCAO PRINCIPAL(){
  INTEIRO limite;
  LER(INTEIRO, limite);
  fibonacci(limite);
  RETORNE ;
}
```

- Shell Sort

```
INTEIRO FUNCAO shell(INTEIRO array[], INTEIRO tamanho){
  INTEIRO i;
  INTEIRO j;
  INTEIRO valor;
  INTEIRO h;

  ITERADOR(h = 1, h < tamanho, h = h * 3 + 1){
     #so alterando o h
  }

  ITERADOR( h = h/3 , h < 1 , ){
     ITERADOR(i = h, i < tamanho, i = i + 1){
        valor = array[i];
        ITERADOR( j = i  - h , j >= 0 E valor < array[j] , j = j  - h){
           array[j + h] = array[j];
        }

        array[j+h] = valor;
     }
  }

  RETORNE array;
}

VAZIO FUNCAO PRINCIPAL(){
  INTEIRO vetor[10];
  INTEIRO saida[10];

  vetor = [9, 8, 3, 2, 5, 1, 4, 7, 6, 0];

  saida = shell(vetor, 10);
}
```

Especificação do Interpretador:

- Interpretador criado em Python 3

1. A Enumeração foi feita com de variáveis de tuplas

```
 NAO_RECONHECIDO = (0, 'nao reconhecido')
VARIAVEL = (1, 'variavel')
IGUALADOR = (2, 'igualador')
MAIOR_QUE = (3, 'maior que')
MENOR_QUE = (4, 'menor que')
DIFERENTE = (5, 'diferente')
MAIOR_IGUAL = (6, 'maior igual')
MENOR_IGUAL = (7, 'menor igual')
ATRIBUICAO = (8, 'atribuicao')
NUMERO = (9, 'tipo: numero')
CARACTERE = (10, 'tipo: caractere')
CARACTERES = (11, 'tipo: caracteres')
MAIS = (12,'mais')
MENOS = (13, 'menos')
VEZES = (14, 'vezes')
DIVIDIR = (15, 'dividir')
PARENTESES_ESQ = (16, 'parenteses esquerdo')
PARENTESES_DIR = (17, 'parenteses direito')
CHAVE_ESQ = (18, 'chave esquerda')
CHAVE_DIR = (19, 'chave direita')
CONCHETE_DIR = (20, 'conchete direito')
CONCHETE_ESQ = (21, 'conchete esquerdo')
VIRGULA = (22, 'virgula')
PONTO_VIR = (23, 'ponto e virgula')
ASPAS = (24, 'aspas')
ASPAS_DU = (25, 'aspas duplas')
CONCATENACAO = (26, 'concatenacao')
FUNCAO = (27, 'funcao')
PRINCIPAL = (28, 'principal')
LER = (29, 'leitura')
IMPRIMIR = (30, 'escrita')
RETORNE = (31, 'retorno')
SE = (32, 'condicional se')
MAS_SE = (33, 'condicional mas se')
SENAO = (34, 'condicional senao')
FAZER = (35, 'repeticao faz')
ENQUANTO = (36, 'repeticao enquanto')
ITERADOR = (37, 'repeticao iterador')
BOOLEANO = (38, 'tipo: booleano')
NAO = (39, 'logico nao')
E = (40, 'logico e')
OU = (41, 'logico ou')
VAZIO = (42, 'tipo: vazio')
INTEIRO = (43, 'tipo: inteiro')
BRANCO = (44, 'branco')
COMENTARIO = (45, 'comentario')
```

2. Tabela de expressões regulares feita com dicionário em python 3

regras = {

```
    '[\'|\"]\S[\'|\"]' : CARACTERE,
    '[\'\"].*?[\'\"]' : CARACTERES,
    '[a-zA-Z][a-zA-Z0-9_]*' : VARIAVEL,
    '\==' : IGUALADOR,
    '\>' : MAIOR_QUE,
    '\<' : MENOR_QUE,
    '\!=' : DIFERENTE,
    '\>=' : MAIOR_IGUAL,
    '\<=' : MENOR_IGUAL,
    '\=' : ATRIBUICAO,
    '\d+' : NUMERO,
    '\+' : MAIS,
    '\-' : MENOS,
    '\*' : VEZES,
    '\/' : DIVIDIR,
    '\(' : PARENTESES_ESQ,
    '\)' : PARENTESES_DIR,
    '\{' : CHAVE_ESQ,
    '\}' : CHAVE_DIR,
    '\[' : CONCHETE_DIR,
    '\]' : CONCHETE_ESQ,
    '\,' : VIRGULA,
    '\;' : PONTO_VIR,
    '\'' : ASPAS,
    '\"' : ASPAS_DU,
    '\&' : CONCATENACAO,
    '[\#].*?[\#]' : COMENTARIO
```

}

3. Tabela de palavras reservadas feita com dicionário em pyhotn3

palavras_reservadas = {

```
    'FUNCAO' : FUNCAO,
    'PRINCIPAL' : PRINCIPAL,
    'LER' : LER,
    'IMPRIMIR' : IMPRIMIR,
    'RETORNE' : RETORNE,
    'SE' : SE,
    'MAS_SE' : MAS_SE,
    'SENAO' : SENAO,
    'FAZER' : FAZER,
    'ENQUANTO' : ENQUANTO,
    'ITERADOR' : ITERADOR,
    'VERDADEIRO' : BOOLEANO,
    'FALSO' : BOOLEANO,
    'NAO' : NAO,
    'E' : E,
    'OU' : OU,
    'VAZIO' : VAZIO,
    'INTEIRO' : INTEIRO,
```

}