Especificação da Linguagem

POGLANG

Jonas Edward Tashiro
Luan Lopes Barbosa de Almeida
Rafael Melloni Chacon Arnone
Mateus Assalti Santana

Contents

1 Introdução

PogLang é uma linguagem procedural que possui inspiração nas linguagens C, GO e Rust; é fortemente tipada, estaticamente tipada e não possui um garbage colector (diferente de GO).

O objetivo do PogLang é ser uma linguagem educativa, mostrando o processo de criação de um Compilador com capacidade de compilar tanto para processadores x86 quanto para o bytecode do WebAssembly.

2 Variáveis e Operadores

2.1 Declarações e Tipos

Na linguagem PogLang todas as variáveis devem ser declaradas antes de serem usadas, isto é, o tipo deve indicado de forma explicita junto ao nome da variável, sendo que o tipo é escrito primeiro seguido do nome da variável.

Também é possível declarar várias variáveis do mesmo tipo, na qual, após escrevermos o tipo da variável colocamos uma lista com o nome das variáveis separadas por ’,’.

Listing 1: Exemplo declaração de variáveis

```
1 int pog;
2 int x, y, z;
```

Outra forma de declararmos uma variável é através da palavra-chave let, na qual inicializamos uma variável com um valor definido. Neste caso não é necessário declarar o tipo da

variável, pois este é inferido a partir do literal a direita do símbolo de igualdade '='.

Listing 2: Exemplo inicialização com palavra-chave let

```
1 let x = 2; //declara e inicia uma variavel do tipo int
2 let f = 3.14f //declara e inicia uma variavel do tipo float
3 let a = [1,2,3] //declara e inicia um array do tipo int
```

Na questão do tipo de variáveis, a linguagem PogLang possui os seguintes tipos apresentados na tabela abaixo:

Type	Bytes	Intervalo
char	8	0 até 255
int	32	–32,767 até 32,767
uint	32	0 até 65,535
float	32	$-3.4 \cdot 10^{38}$ até $+3.4 \cdot 10^{38}$
double	64	$-1.8 \cdot 10^{308}$ até $+1.8 \cdot 10^{308}$
bool	32	-

A linguagem PogLang possui vários tipos de operadores, sendo que estes podem ser divididos nas seguintes classes: aritméticos, relacionais e lógicos, sendo que, o único operador que fica separado destas classes e o operador de atribuição e é este que analisamos em seguida.

2.2 Operador: Atribuição

O operador de atribuição e uma relação binaria que copia o valor de uma expressão no local do operando da direita para uma variável na posição de operando esquerdo, sendo que ambos

os operandos necessitam ser do mesmo tipo.

Portanto a linguagem PogLang segue a mesma definição de atribuição da linguagem C.

Listing 3: Exemplos de Atribuição

```
let y = 2;
int x;
x = (3 + 5) * y;
```

2.3 Operadores Aritméticos

A tabela a seguir mostra os operadores aritméticos e seus efeitos:

Operador	Ação
-	Subtração, menos unário
+	Adição
*	Multiplição
/	Divisão
%	Resto de Divisão

2.4 Operadores Relacionais

A tabela a seguir mostra os operadores relacionais e seus efeitos:

Operador	Ação
>	Maior que
>=	Maior que ou igual ($\geq$)
<	Menor que
<=	Menor que ou igual ($\leq$)
=	Igual
!=	Não igual($\neq$)

2.5 Operadores Lógicos

A tabela a seguir mostra os operadores lógicos e seus efeitos:

Operador	Ação
&&	AND
||	OR
!	NOT

2.6 Precedência de Operadores

Uma parte importante sobre operadores e a questão de precedência, na qual os certos operadores tem prioridade ao serem encontrados em uma expressão, i.e., são realizados primeiro. A questão da precedência, se não entendida, pode gerar expressões ambíguas, portanto a tabela a seguir e fornecida.

Maior	() []
	!
	* / %
	+ -
	<<= >>=
	== !=
	&&
Menor	||

3 Literais

Literais são valores constantes para certos tipos de dado que são representados por certas cadeias de caracteres.

Há seis literais no PogLang:

- Integer literal
- Float literal
- Double literal
- Character literal
- String literal
- Boolean literal

O *Integer literal* representa apenas valores inteiros, sendo que este é escrito na base decimal. Por exemplo, 45, 69, 420, etc.

O *Float literal* representa números reis. Sua representação e particionada em duas partes, separadas pelo caractere '.', a parte inteira e a parte fracionada. Do mesmo modo que representamos *Integer literal* em base decimal, fazemos o mesmo *Float literal* com a adição de um sufixo que permite a diferenciação de um *Double literal*, que segue as mesmas especificações.

- f: especifica o literal como do tipo float. Por exemplo, $4.0f$, $32.7f$, etc.
- lf: especifica o literal como do tipo double. Por exmplo, $1024.0lf$, $128.89lf$, etc.

O *Character literal* é apenas um caractere delimitado dentro de aspas simples. Por exmplo, 'a', 'b', etc.

O *String literal* é uma cadeia de caracteres delimitada dentro de aspas dulplas. Por exemplo, "Hello", "Fuba", etc.

O *Boolean literal* possui apenas dois valores, true e false.

4 Instruções de Repetição

Em uma linguagem de programação as instruções de repetição controlam a quantidade de vezes que um bloco será executado baseado em uma condição, essas instruções são também chamadas de loops sendo alguns exemplos as instruções de while e for e do-while.

Listing 4: Forma geral do while

```
1 while expression
2 {
3     statements
```

```
}
```

Listing 5: Exemplo while

```
int i = 0;
while i < 5
{
    print(i);
    i = i + 1;
}
```

Neste exemplo a saída no terminal possui números de 0 até 4.

A instrução while representa um loop onde, enquanto a expressão for verdadeira nos executamos o statement.

Listing 6: Forma geral do for

```
for i in 0..n
{
    statements
}
```

Um exemplo dessa instrução seria:

Listing 7: Exemplo for

```
for i in 0..5
{
    print(i)
}
```

Neste exemplo a saída no terminal e a mesma do Exemplo while.

A instrução for representa um loop cujo funcionamento

pode ser compreendido atraves da frase "para i de um 0 até n-1 execute um determinado bloco".

Listing 8: Forma geral do do-while

```
do
{
    statements
} while expression;
```

Listing 9: Exemplos do-while

```
int i = 0;
do
{
    print(i);
    i = i + 1;
} while i < 5;
```

A instrução do-while representa um loop que já começa executando determinado statement e verifica a expressão para realizar a execução do statement novamente após já ter executado ele uma vez.

5 Instrucoes de Controle

Em uma linguagem de programação as instruções de controle controlam a ordem de execução das instruções com base em uma condição,

Na linguagem Poglang um ";" indica o final de uma expressão, nessa linguagem as chaves "{" e "}" são utilizadas para agrupar declarações e expressões em um bloco, uma observação e que não precisamos adicionar ";" após um "}" para indicar o

final do bloco.

A instrução if-else e utilizada para representar uma decisão, podendo ser representada através deste trecho, na qual o else e opcional:

Listing 10: Forma geral if-else

```
1 if expression {
2    statement1
3 } else {
4    statement2
5 }
```

Quando avaliada a expression (tipo booleano) for avaliada, se for verdadeira (true) o statement1 e executado, caso contrário se o else existir executamos o statement2.

Listing 11: Forma Geral de múltiplos if-else

```
1 if expression {
2     statement
3 } else if expression {
4     statement
5 } else {
6     statement
7 }
```

Outro constructo suportado pela Poglang e o else-if que representa uma decisão com múltiplos caminhos, as expressões são avaliadas em ordem, se essa expressão for verdadeira então o statement relacionado a ela será executado terminando com a cadeia de decisão, nesse caso podemos utilizar o else para representar todas as outras alternativas.

Para podermos ilustrar um exemplo utilizando instruções de controle vamos demonstrar o exemplo de busca binaria, em que o array itens está ordenado e x representa o item que desejamos encontrar.

Listing 12: Exemplo if-else

```
fn binsearch(int x, int[] items) -> int
{
        int low, high, mid;
        low = 0;
        high = items.len() - 1;
        while low <= high
        {
                mid = (low + high)/2;
                if x < items[mid] {
                        high = mid + 1;
                } else if x > items[mid] {
                        low = mid + 1;
                } else {
                        return mid;
                }
        }
        return -1;
}
```

6 Arrays

Arrays em uma linguagem de programação são espaços contíguos na memoria do computador, na linguagem Poglang temos um diferencial da linguagem C, o armazenamento do tamanho do array, facilitando o uso arrays.

Podemos instanciar um array através da seguinte expressão:

Listing 13: Forma geral do array

```
type[size] name;
```

Em que type representa o tipo da variável que será armazenado em nosso espaço de memória contiguo e size o tamanho desse espaço e finalmente name que representa o nome da nossa variável.

Listing 14: Exemplo de declaração de um array

```
int[4] array;
```

Podemos utilizar o método len no nosso array para descobrir seu tamanho, nesse caso array.len() resultaria em 4.

O exemplo abaixo mostra como utilizar o método len para mostrarmos na tela números de 0 até 3:

Listing 15: Exemplo uso do for para um array

```
int[4] array;
for i in 0..array.len()
{
```

```
4     print(i);
5 }
```

Podemos inicializar um array com itens através da sintaxe, na qual esses itens precisam ser do mesmo tipo de separados por vírgula:

Listing 16: Exemplo inicialização de array com let

```
1 let nome = [item1, item2,...itemN];
```

Podemos, ainda assim, determinar o tamanho do array mesmo que este não tenha sido declarado de maneira explicita ao utilizar a inicialização acima

Listing 17: Exemplo do .len()

```
1 let array = [1, 2, 3, 4];
2 for i in 0..array.len()
3 {
4     print(array[i]);
5 }
```

7 String

Na linguagem Poglang as strings são uma cadeia de caracteres guardados em um array de chars. Diferente da linguagem C não utilizamos um caractere para delimitar o fim da string, uma vez que, arrays conseguem utilizar o método len para retornar o seu tamanho.

O exemplo abaixo demonstra o uso de string para mostrar na tela a palavra "pog":

Listing 18: Exemplo string

```
char[3] name = ['p', 'o', 'g'];
for i in 0..name.len()
{
    print(name[i]);
}
```

8 Funções

Funções provem um modo conveniente de encapsular e modularizar alguma computação, na linguagem PogLang este importante conceito se inspirou nas linguagens Rust e GO

O próximo exemplo demostra um exemplo de uma função modelando a ideia matemática dá função de exponenciação em números inteiros.

Listing 19: Exemplo função power

```
fn power(int m, int n) -> int
{
  for i in 1..=n
  {
    m = m * m;
  }
  return m;
}
```

Na primeira linha, a função é identificada pela palavra reservada fn e ao lado temos o nome da função, nomes e tipos dos parâmetros e após a flecha "->" o tipo do retorno da função.

Os parâmetros são colocados entre os parênteses e são separados pelo símbolo ',', sendo que, estas variáveis são locais a função, i.e., não são visíveis para demais regiões do código, somente no escopo da função.

O valor que power calcular é retornado através da palavra-reservada return, na qual, o retorno de uma função pode ser qualquer um dos tipos definidos na linguagem e vazio através da palavra reservada void.

Para uma definição mais geral de função, estas tem a seguinte forma:

Listing 20: Forma geral de funções

```
1 fn funtion-name(parameter declarations) -> return-type
2 {
3   statements
4   return value-to-return;
5 }
```

9 Escopo de Variáveis

A questão de escopo de variáveis na linguagem PogLang segue o padrão já conhecido por linguagens como Java e C, na qual o escopo está associado ao bloco de código onde a variável é declarada, portanto esta será visível no bloco em que foi declarada e para todos os blocos que contém o bloco a qual

ela pertence.

Além disso, quando o fim do bloco for alcançado (ao alcançar o símbolo '}') a variável é descartada do stack, sendo assim impossível de utilizá-la nas próximas linhas do código.

Listing 21: Exemplo de escopo

```
{
  int x;
  {
    int y; //visivel para o bloco de fora
  }
  y = 2; // resulta em erro
}
```

10 Referências

1. KERNIGHAN B. W.; RITCHIE D. M.; C Programming Language. 2. ed. Estados Unidos: Pearson, 1988. 5p.

2. SCHILDT H.; Java: The Complete Reference. 12. ed. Estados Unidos: McGraw Hill, 2021. 45p.

3. SCHILDT H.; C++ : The Complete Reference. 4 ed. Estados Unidos: McGraw Hill, 2002. 14p.

4. BLANDY J.; ORENDORF J.; TINDALL L. F. S.; Programming Rust Fast, Safe Systems Development. 2. ed. Estados Unidos: O'Reilly Media, 2021. 79p.

5. DONOVAN A.; KERNIGHAN B. W.; Go Programming Language. 1. ed. Estados Unidos: Addison-Wesley Professional Computing, 2015. 27p.