

Universidade Federal de Alagoas

Instituto de Computação

Ciência da Computação

Compiladores

2018.1

Nameless-lang Especificações

Lucas Antonio Ferro do Amaral

Nelson Douglas C. Oliveira

13/07/2018

1 Estrutura do código	2
2 Tipos de dados	2
2.1 Identificadores	2
2.2 Inteiro	2
2.3 Ponto flutuante	3
2.4 Caractere	3
2.5 Vetores unidimensionais	3
2.6 Strings(Cadeia de Caracteres)	4
2.7 Booleano	4
3 Constantes	5
4 Operadores	5
4.1 Atribuições	5
4.2 Aritméticos	5
4.3 Relacionais	6
4.4 Lógicos	6
4.5 Caracteres e cadeias de caracteres	7
4.5.1 Concatenação	7
5 Instruções	7
5.1 Condicional de um via	7
5.2 Condicional de duas vias	8
5.3 Estrutura iterativa de controle lógico	8
5.4 Funções	9
5.5 Exemplos de código	10
5.5.1 Hello World	10
5.5.2 Fibonacci	10
5.5.3 Shell sort	11
5.6 Expressões regulares	13

1 Estrutura do código

- Variáveis podem ser declaradas em qualquer linha de código, e dentro do inicializador do escopo do for
 - Pode ser feito

```
...  
int a = 1  
if(a == 1)  
{  
    int b = 10 # variável apenas vista no  
               # escopo do if  
    #block of code here  
}  
...
```

2 Tipos de dados

- A Linguagem é sensível a letras maiúsculas e minúsculas

2.1 Identificadores

Os identificadores são da forma:

- Inicia-se com letra maiúscula ou minúscula;
- Os seguintes caracteres podem ser letras, números ou underlines;
- Não podem ser usado espaços em branco e caracteres especiais;
- O tamanho do identificador é ilimitado.
- A linguagem é fortemente tipada

2.2 Inteiro

- Representa um número inteiro e seus literais representam um número deste tipo;
- São declarados pela palavra reservada 'Int' seguida de um identificador e uma sequência de dígitos sendo obrigado a ter o primeiro dígito e terminando em ';'.
- Exemplo:

```
int x
```

```
x=10
```

2.3 Ponto flutuante

- Seus literais representam um número real seguida de um identificador;
- Declarado pela palavra reservada 'float';
- A parte inteira dos literais é separada da parte decimal por um ponto “.”;
- Exemplo:

```
float x  
x=3.14
```

2.4 Caractere

- Declarado pela palavra reservada 'char' seguida de um identificador;
- Seus literais representam um caractere codificado em ASCII;
- Seus literais são representados entre aspas simples;
 - Exemplo

```
char myChar  
myChar = 'x'
```

2.5 Vetores unidimensionais

- São declarados com o tipo utilizado no vetor seguido de '::' e o tamanho máximo do vetor;
- Representam listas sequenciais de um mesmo tipo;
- O tamanho mínimo desta lista é zero, ou seja, uma lista vazia;
- Seus itens podem ser acessado especificando no identificador o seu índice da seguinte forma:
 - id::index;
 - onde id é o identificador;
 - e index é um inteiro maior ou igual a zero;
- O primeiro elemento de um vetor está sempre no índice zero;
- Não são permitidos índices negativos;
- E 'id.size' informa o tamanho que esse vetor possui
- Exemplo:

-

```
int vec::5
vec::1 = 5
vec::2 = 350
print(vec.size)
```

2.6 Strings(Cadeia de Caracteres)

- Declarado pela palavra reservada 'string' seguida de um identificador;
- São construídas como um vetor de caracteres
- Seus literais são representados entre aspas duplas;
- String igual vetor possui um atributo para informar o seu tamanho
- Exemplo

```
string str::10
str = "ayyy lmao"
print(str.size) # == 10
```

2.7 Booleano

- São declarados com a palavra reservada 'Bool' seguida de um identificador;
- Booleanos representam dois únicos possíveis estados lógicos, TRUE ou FALSE;
- Estes estados também podem ser representados, respectivamente, com os inteiros 1 ou 0;
- Exemplo:

```
bool x
x = true
x = false
x = 1
x = 0
```

3 Constantes

- Inteiro
 - sequência de dígitos

- Caractere
 - uma aspa simples seguido do caracter e termina com uma aspa simples
 - ex: 'c'
- vetor unidimensional
 - caracter '[' seguido dos elementos separados por vírgula e terminando com ']'
 - ['a', 'b', 'c', 'd'] vetor de caracteres
 - [1, 2, 3, 4] vetores de inteiros
 - [1.0, 2.0, 3.0, 4.0] vetor de ponto flutuantes
- Ponto Flutuante
 - dígito seguido de um '.' e outro dígito.
 - 1.0
 - 2.3
 - 3.14

4 Operadores

4.1 Atribuições

A atribuição é feita com o operador "=", com a seguinte sintaxe:

Do lado esquerdo fica o ID da variável a receber a atribuição

Do lado direito fica o valor a ser armazenado. O valor deve ser do mesmo tipo do ID, pois a linguagem não faz coerção.

```
int valor = 1
```

4.2 Aritméticos

- "+": Soma
- "-": Diferença
- "*": Multiplicação
- "/": Divisão
- O operador aritmético unário negativo é "~".
- Procedência dos operadores aritmeticos:
 - '*' e '/'
 - '+' e '-'

4.3 Relacionais

Estes são os operadores relacionais suportados para tipos numéricos:

Os operadores "==" e "!=" também podem ser usados com booleanos

- "==": Igualdade entre dois operandos

- “!=”: Desigualdade
- “<”: Menor que
- “>”: Maior que
- “<=” Menor ou igual que
- “>=” maior ou igual que

```
if (1 <= 2) {
    #Faça algo
};
```

4.4 Lógicos

Os três operadores lógicos suportados são

- “not”: um operador unário que nega uma expressão lógica.
- “and”: Operador binário que faz um “and” lógico entre duas expressões
- “or”: Operador que faz um “ou” lógico entre duas expressões
- Precedência dos operadores Lógicos segue abaixo:
 - and
 - or
 - not

```
if (a or b) {
    #faça algo
}
```

4.5 Caracteres e cadeias de caracteres

- “\$”: Concatenação

4.5.1 Concatenação

- A concatenação de caracteres ou cadeias de caracteres é feita com o operador sobrecarregado “\$”
- Uma concatenação sempre retorna uma cadeia de caracteres, independente dos tipos usados

```
int x
x = 10
char str::2
str = "AB"

x$str
#retorna "10AB"

char c
c = 'A'
c$x
#Retorna "A10"
```

5 Instruções

Os parâmetros das instruções são colocados entre parênteses e separados por vírgula. O escopo é definido entre chaves.

5.1 Condicional de um via

- **if**

A instrução condicional if é relacionada a uma variável ou expressão booleana, a qual é definida entre parênteses.

O escopo do if é definido entre chaves e fechado por ponto e vírgula. Este escopo apenas é executado caso a expressão booleana entre os parênteses seja True, caso contrário o compilador ignora código dentro das chaves.

```
if (true){
    #Faça algo
}
```

5.2 Condicional de duas vias

- **if-else**

A instrução if-else, assim como a if, também é relacionada a uma expressão ou variável booleana. O diferencial é que esta instrução têm dois possíveis escopos, um para o if e outro para o else. Caso a expressão booleana nos parênteses seja True, o escopo do if é

executado e o do else ignorado. Caso a expressão seja False, o escopo do if é ignorado e o do else é executado.

```
if (false){  
    #Faça algo 1  
} else {  
    #Faça algo 2  
}
```

Vale notar que por se tratar de uma única expressão, o ponto e vírgula vem após a chave de fechamento do else e não após a do if, como é feito na instrução de uma via.

5.3 Estrutura iterativa de controle lógico

- **while**

Esta instrução é controlada por uma expressão lógica a qual é escrita entre parênteses. O escopo da instrução é definido entre chaves e, como em todas instruções, é fechado por ponto e vírgula. O while continua repetindo as instruções de seu escopo enquanto a expressão lógica que o controla for True. Se em algum momento esta expressão se tornar False, o loop para e a execução da instrução termina.

```
while (True){  
    #faça algo  
}
```

Estrutura de controle iterativo controlada por contador

- **for**

A estrutura for recebe três parâmetros, índice, passo e limite. Ela executa as instruções em seu escopo enquanto o índice for menor que o limite definido. A cada iteração o seu índice será incrementado em uma unidade do passo, i.e. índice+passo.

```
for (int x=0,1,10){  
    #Faça algo 10 vezes  
}
```

5.4 Funções

As funções precisam especificar o seu tipo de retorno. Funções são definidas pela palavra reservada 'function', seguida dos parâmetros entre parênteses, e a definição da

função entre chaves. O retorno da função é feito com a palavra 'return' seguida de o valor a ser retornado. Caso não seja seguida de nada na mesma linha, 'return' para a execução da função.

Os parâmetros são passados como cópia e representam uma variável local internamente nas funções.

```
int function sum(int x, int y){  
    return x+y  
}
```

Para executar a função em uma entrada, usamos o nome da função com os parâmetros a serem processados entre parênteses.

```
sum(1,2) #retorna 3
```

5.5 Exemplos de código

5.5.1 Hello World

```
int main(int argc, string argv)  
{  
    print(" Hello World")  
}
```

5.5.2 Fibonacci

```
void fibonacci(int n)  
{  
    int sequence::n
```

```

sequence::0 = 1

if (n==1){
    print(1)
    return
}

if (n>=2){
    print(1)
    sequence::0 = 1
    sequence::1 = 2
}

if (n==2){
    print(",2")
    return
}
int i=2
while (i<n){
    sequence::i = sequence::(i-1)+sequence::(i-2)
    print(",")
    print(sequence::i)
}
}

```

5.5.3 Shell sort

```

int:: shellSort(int nums::, int n)
{
    int h = 1
    while(h < n)
    {
        h = h * 3 + 1
    }
    h = h / 3

```

```

    int c, j

    while (h > 0)
    {
        for (int i = h, i < n, i++)
        {
            c = nums::i
            j = i
            while (j >= h && nums[j - h] > c) {
                nums::j = nums::(j - h)
                j = j - h
            }
            nums::j = c
        }
        h = h / 2
    }
    return nums
}

```

6. Especificação dos Tokens

6.1 Linguagem de programação da implementação

- Julia
- <https://julialang.org/>

6.2 Enumeração das Categorias E Tokens

```
{
```

"EOF" : -1,
"CT_VALUE": 1,
"CT_INT": 2,
"CT_CHAR": 3,
"CT_FLOAT": 4,
"CT_VEC": 5,
"SMCL": 6,
"EPS": 7,
"ID": 8,
"CONST": 9,
"EXPR": 10,
"OPR_PM": 11,
"OPR_DM": 12,
"OPRLN": 13,
"OPRLR_EQ": 14,
"OPRLR_LG": 15,
"OPRLR_LGEQ": 16,
"FN_PRINT": 17,
"FN_READ": 18,
"DREAD": 19,
"BLK_IF": 20,
"BLK_ELS": 21,
"BLK_FOR": 22,
"BLK_WHILE": 23,
"COMMA": 24,
"O_BRCKT": 25,
"C_BRCKT": 26,
"O_C_BRCKT": 27,
"C_C_BRCKT": 28,
"O_PRTSIS": 29,
"C_PRTSIS": 30,
"OPR_ATR": 31,
"IDT_INT": 32,
"IDT_FLOAT": 33,
"IDT_CHAR": 34,
"IDT_STRING": 35,
"FN_MAIN": 36,
"VEC_IN": 37,
"LEX_ERR": 38,
"OPR_SOM": 39,
"OPR_SUB": 40,
"CTN": 41,

```

"PARAMS": 42,
"IDT_BOOL": 43,
"EXP_BOOL": 44,
"VOID" : 45,
"FN_DCLR": 46,
"FN_CALL": 47,
"CT_STRING": 48,
"OPR_CONCAT": 49,
"CONST_KEY": 50,
}

```

5.6 Expressões regulares

```

opr_pm = '+|-'
opr_dm = '/|*'

```

```

oprlr_eq_dif = '==|!='
oprlr_lgt = '<|>|>=|<='
opr_concat = '\+\\++'
ct_str = '".*?({opr_concat}{ct_string}?{ct_str})'
comma = ","
o_bracket = '['
c_bracket = ']'

```

```

opr_and_or = 'and|or'

```

```

opr_not = 'not'

```

```

oprlr = '{oprlr_eq_dif} | {oprlr_lgt} | {oprlr_lgt_eq} | {opr_and_or}'

```

```

ct_str = '\".*\"$
ct_int = '{opr_pm}?[0-9]+$'
ct_float = '{ct_int}.[0-9]*$'
ctn = '{ct_int}|{ct_float}'
id = '[:alpha:][:alnum:]*'

```

```

loop_while = 'while [[:blank:]]+\\({expr_bool}\\)[[:blank:]]+ \\{cmd\\}'

```

```

loop_for = 'for[[:blank:]]+\\({int_dclr},{ct_int},{ct_int}\\)[[:blank:]]*
\\{cmd\\}'

```

ctb = '1 | 0 | true | false'

expr_bool = '{expr_arit} {opr_logic} {expr_arit} | {ctb}'

term = '{ctn}|{id}'

expr_arit = '{term} {opr_algebc} {term} | \({expr_arit}
({opr_algebc}{expr_arit})*\)({opr_algebc}{expr_arit})*'

data = '{expr_arit} | {term} | {expr_bool}'

int_dclr = 'int {id} = [0-9]*'

float_type = 'float'

bool_type = 'bool'

char_type = 'char'

string_type = 'string'

void_type = 'void'

data_type = '{float_type} | {bool_type} | {char_type} | {string_type} | {void}'

var_dclr = '{int_dclr} | ({bool_type} | {float_type} | {char_type} | {string_type})
{atrib}'

atrib = '{id} = {data}'

cmd = '{loop} | {atrib} | {expr_arit} | {expr_bool} | {term}'

params = '{id}|{id}{comma}{id}?({comma}{params})'

fn_dclr = '{data_type} {id}\({params}\)\{\{cmd\}*?(return+{data})\}'

fn_call = '{id}\(?{params}\)'

comment_block = '*[:alnum:]* */ | /*{comment_block}*/'

coment_line = '#[:alnum:]*'

comment = 'comment_block | comment_line'