UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
COMP263 - Compiladores

LINGUAGEM DUMA

Eduarda Tatiane Caetano Chagas
Marcos Gleysson Silva do Nascimento

Maceió-AL
2017.1

Sumário

CAPÍTULO 1

ESPECIFICAÇÃO MÍNIMA DA LINGUAGEM

1.1 Introdução

Tal relatório possui o objetivo de informar brevemente algumas informações a cerca da especificação da linguagem DUMA, que foi formulada com o objetivo da fixação de conceitos básicos sobre programação, construções de linguagens e compiladores.

A linguagem DUMA compreende uma linguagem de programação compilada, estruturada, imperativa e procedural cujo domínio de aplicação constitui a implementação de algoritmos e aplicações simples.

1.2 Estrutura geral de um programa

A linguagem DUMA consiste em uma linguagem de alto nível onde as suas instruções são divididas em módulos, denominados **funções**, que são definidos a partir de um **identificador**, que deve ser único, isto é, diferente dos demais identificadores de funções ou variáveis, e de um **tipo de retorno**, que pode ser: *inanis* (vazio), *integer* (inteiro), *realem* (ponto flutuante), *boolean* (booleano), *litterae* (caractere) ou *sermo* (string).

Como pode ser observado acima, a linguagem DUMA possui suas palavras reservadas retiradas do latim, antiga língua indo-europeia do ramo itálico.

O ínicio do programa sempre ocorrerá com a chamada a função ***initium***.

Os comentários, que serão sempre de linhas, são caracterizados por inicializarem sempre com o símbolo #. Os delimitadores de escopo serão representados pelos símbolos { e } . E cada final de instrução deve ser representado por um **.** (ponto final).

Outra regra a ser seguida trata-se da extensão do código escrito, que deverá ser

possuir sempre a extensão .duma para que possa ser compilado corretamente.

Tabela 1: Estrutura geral de um programa em DUMA

```
duma <nome do programa>
# declaração de constantes
const {
<nome da constante 1> = valor 1;
<nome da constante n> = valor n;
}

# declaração de variáveis
var {
<tipo da variável> var1, var2, ..., varN;
}

# declaração de funções
fun {
<tipo de retorno> <nome da função> (<tipo1> <nome1>, ... , <tipoN> <nomeN>);
}

inanis initium(){ #Função de inicialização

    Declarações.
    Comandos.

    <tipo de retorno> função1(declaração de parâmetros){ # Exemplo função 1

        Declarações locais.
        Comandos.

    }
    ...
    <tipo de retorno> funçãoN(declaração de parâmetros){ # Exemplo função N

        Declarações locais.
        Comandos.

    }
}
```

1.3 Características léxicas

Existem algumas regras a serem seguidas durante a nomeação de novas funções e variáveis em DUMA. Os nomes ou identificadores devem exclusivamente começarem com letras, sendo no entanto permitido o uso de outras letras, dígitos ou underscore ao longo deste. Porém, será apontado um erro quando o identificador for composto por caracteres especiais (Ex: !, @, #, $, %, &, , (,),{, }, ?, [,], /).

O nome do programa digitado logo após a palavra *duma* no início do programa não deve obrigatoriamente ser igual ao nome do arquivo, mas segue a mesma regra de escrita de nomes descritas anteriormente.

O limite máximo de tamanho de um identificador em DUMA são de 16 caracteres e também não será aceito qualquer nomeação que coincida com alguma palavra reservada.

Para que o usuário não venha a declarar alguma variável ou função com alguma palavra reservada, segue abaixo uma relação com todas estas.

Tabela 2: Conjunto de palavras reservadas em DUMA

Funções	initium
Tipos	inanis, integer, realem, boolean, litterae e sermo
Comandos de seleção	si (if), aliud (else) e sialiud (elseif)
Comandos de interação	quia (for) e dum (while)
Operadores lógicos	verum (true) e falsus (false)
Outras palavras reservadas	duma, var, fun, const

1.4 Especificação dos tipos

As variáveis de DUMA são todas **estaticamente tipadas**, isto é, elas devem ser declaradas antes de sua utilização, sendo neste momento definido o seu tipo que irá permanecer inalterado até o final do período de vida desta, não havendo nesta linguagem nenhum tipo de mecanismo de coerção.

Em DUMA todas as suas declarações de varíaveis e de constantes devem estar dentro dos seus respectivos blocos de declarações, todas as suas declarações possuem a seguinte forma:

```
<tipo> <identificador>;
```

Uma regra muito importante segue em relação a declaração ded variáveis que não se pode realizar atribuições. TOdos as variáveis são inicializadas com os seguintes valores default mostrados na Tabela 3.

- *integer* : inteiro de 32 bits, seus literais são expressos em uma sequência de digitos decimais.
- *realem* : ponto flutuante de 32 bits, seus literais são expressos em uma sequência de digitos decimais seguidos de um ponto e mais digitos.
- *litterae* : caracteres da tabela ascii, seus literais são representados por caracteres entre apóstrofos ' '.
- *boolean* : existem dois valores deste tipo, true e false.
- *sermo* : cadeia de caracteres cujos literais são representados entre aspas duplas.
- *matrix <tipo>[tamanho] <identificador>* : arranjo unidimensional

Tabela 3: Valores default

integer	0
realem	0.0
litterae	' '
boolean	false
sermo	null
matrix	null

1.4.1 Inteiro

- Palavra reservada: **integer**
- Operações permitidas:
  - Lógicas:Não permitido.

  - Aritméticas: Adição, subtração, multiplicação, divisão e resto.
  - Relacionais: Maior que, menor que, igual a, diferente de, maior ou igual e menor ou igual.

1.4.2 Ponto flutuante

- Palavra reservada: **realem**
- Operações permitidas:
  - Lógicas:Não permitido.
  - Aritméticas: Adição, subtração, multiplicação, divisão e resto.
  - Relacionais: Maior que, menor que, igual a, diferente de, maior ou igual e menor ou igual.

1.4.3 Caractere

- Palavra reservada: **litterae**
- Operações permitidas:
  - Lógicas:Não permitido.
  - Aritméticas: Não permitido.
  - Relacionais: Maior que, menor que, igual a e diferente de.
  - Concatenação.

1.4.4 Booleano

- Palavra reservada: **boolean**
- Operações permitidas:
  - Lógicas: Disjunção, conjunção e negação.
  - Aritméticas: Não permitido.
  - Relacionais: Igual a e diferente de.

1.4.5 Cadeia de caracteres

- Palavra reservada: **sermo**
- Operações permitidas:
  - Lógicas:Não permitido
  - Aritméticas: Não permitido.
  - Relacionais: Igual a e diferente de.
  - Concatenação.

1.4.6 Arranjos unidimensionais

- Palavra reservada: **matrix**
- Forma geral: matrix Tipo[quantidade de elementos]
- Operações permitidas:
  - Lógicas: Não permitido.
  - Aritméticas: Não permitido.
  - Relacionais: Não permitido.

1.5 Conjunto de operadores

1.5.1 Aritméticos

Tabela 4: Conjunto de operadores aritméticos

Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto	%

1.5.2 Relacionais

Tabela 5: Conjunto de operadores relacionais

Maior que	>
Menor que	<
Igual	==
Diferente	!=
Maior ou igual que	$\geq$
Menor ou igual que	$\leq$

1.5.3 Lógicos

Tabela 6: Conjunto de operadores lógicos

Negação	!
Conjunção	&&
Disjunção	||

1.5.4 Atribuição

A atribuição na linguagem DUMA consiste em uma instrução. A instrução de atribuição possui o símbolo de igual (=).

1.5.5 Concatenação de cadeias de caracteres

A concatenação de cadeias de caracteres em DUMA será feito por meio do artifício de sobrecarga do operador (.) .

Dessa forma, com apenas este símbolo, podemos unir duas cadeias de caracteres diferentes em apenas uma. Veja este exemplo, que apesar de simples, ilustra exatamente o que acontece na concatenação:

João . zinho = Joãozinho; Passa . tempo = Passatempo; beija . - . flor = beija-flor.

1.5.6 Ordem de precedência e associatividade

A tabela abaixo lista os operadores da linguagem DUMA em ordem de sua precedência (maior para menor). Sua associatividade indica em que ordem operadores de mesma precedência em uma expressão são avaliados.

Tabela 7: Regras de Precedência e Associatividade

Operador	**Descrição**	**Associatividade**
()	Parênteses	esquerda para direita
- !	Menos unário Negação lógica	direita para esquerda
* , / , %	Multiplicação/divisão/módulo	esquerda para direita
+ , -	Adição/subtração	esquerda para direita
<, <=, >, >=	Operadores relacionais	esquerda para direita
==, !=	Igual e Diferente	esquerda para direita
&&	AND lógico	esquerda para direita
||	OR lógico	esquerda para direita
.	Operador de concatenação	esquerda para direita

1.6 Instruções

1.6.1 Estrutura condicional de uma e duas vias

A linguagem DUMA possui os seguintes comandos de seleção:

si (equivalente ao if do C), aliud (equivalente ao else do C) e sialiud (elseif). Abaixo encontram-se a sintaxe básica de cada um destes componentes.

Tabela 8: Estrutura condicional em DUMA

```
# condicional simples apenas com si
si (condição) {

# Conjunto de instruções...

}

# condicional composto com si e aliud
si (condição) {

# Conjunto de instruções I

} aliud {

# Conjunto de instruções II

}

# condicional composto com si , aliud e sialiud
si (condição I) {

# Conjunto de instruções I

} sialiud (condição II) {

# Conjunto de instruções II

} aliud {

# Conjunto de instruções III

}
```

1.6.2 Estrutura iterativa com controle lógico

Em várias situações não se sabe exatamente a quantidade de vezes que um determinado laço deve ser executado, portanto, o processo de controle por contador não é o mais adequado.

Nestes casos, são utilizados os laços controlados logicamente. A continuidade da repetição do laço depende de uma expressão lógica.

As estruturas iterativas com controle lógico em DUMA foram inspiradas na linguagem Java, por meio dos comandos *quia* (equivalente a for), *dum* (equivalente a while) e *facite* (equivalente a do) ... dum. Sendo assim, existem duas estruturas básicas de estruturas desse tipo presentes na linguagem. Ambas estão descritas sintaticamente abaixo.

No caso do comando de repetição *quia* o corpo do laço pode ser uma instrução única; uma instrução composta ou uma instrução nula. A primeira expressão é avaliada uma única vez quando se inicia o loop. Normalmente é utilizada para inicializações. A segunda expressão é o controle lógico do laço, é avaliada antes de cada execução do corpo do mesmo. A terceira expressão é avaliada depois de cada execução do corpo do laço. Normalmente utilizada para incrementos de variáveis.

Tabela 9: Estruturas iterativas com controle lógico em DUMA

```
# Pré-teste
dum (<expressão booleana>) {

    # Conjunto de instruções...

}


# Pós-teste
facite {

    # Conjunto de instruções

} dum (<expressão booleana>);


# semelhante ao for do C
quia (<inicialização> ; <condição> ; <incrementação>) {

    # Conjunto de instruções...

}
```

1.6.3 Estrutura iterativa controlada por contador

Na linguagem DUMA, está presente a seguinte estrutura de iteração controlada por contador semelhante a Python expressa na tabela 10. O primeiro parâmetro, o início, representa o valor inicial, o segundo, representa o valor final e o último representa o tamanho de cada incremento, quando omitido este parâmetro significa que o tamanho do passo é 1 (tamanho padrão).

Tabela 10: Estruturas iterativas com controle por contador em DUMA

```
# estrutura iterativa por contador semelhante a Python
quia <variavel> in spatium(<inicio>,<fim>,<tamanho do passo>){

    # Conjunto de instruções

}
```

1.6.4 Entrada e saída

Em DUMA, utiliza-se os comandos *scribo* e *lectio* para escrita e leitura, respectivamente. O sufixo *ln* pode ser anexado ao comando de escrita, ficando *scriboln* para indicar a escrita seguida de uma quebra de linha (de forma semelhante ao System.out.print e System.out.println presentes no Java). A vírgula utilizada nas operações de escrita é utilizada como um separador para separar o texto fornecido pelo programador dos dados vindos de uma operação de leitura e a operação em questão possui uma lista variável de parâmetros.

Exemplo de operação de escrita:

scribo("Digite os numeros x e y: "); ou
scriboln("Digite os numeros x e y: ");

Exemplo de operação de leitura e escrita correspondente a escrita anterior:

lectio(x,y);
scribo("Os numeros digitados foram: ", x, y)

1.7 Funções

As assinaturas de cada função a ser utilizada no programa deve ser declaradas logo em cima no bloco de funções chamado *fun* ilustrado na tabela 1. Uma função escrita na linguagem Duma é composta por um tipo de retorno, um identificador, uma lista de parâmetros e um par de chaves.

O tipo de retorno pode ser cada um dos tipos existentes na linguagem que foram

expressos anteriormente. A palavra especial *reditus* é utilizada para retornar o valor da função. Se uma função não retornar nada a mesma conterá em seu campo de tipo de retorno a palavra *inanis* que é semelhante ao *void* em C.

Portanto, em DUMA, as funções são declaradas como se segue abaixo na tabela 11.

Tabela 11: Estrutura geral de declaração de funções em DUMA

<tipo de retorno> <identificador> (<lista de parâmetros>){

Conjunto de instruções

}

Exemplos:

inanis enviar_mensagem(integer x){

    scriboln("Oi! O número digitado é: ", x);

}

realem media(realem x, realem y){

    reditus (x+y)/2;

}

1.8 Exemplos práticos

1.8.1 Alô mundo

```
duma hello_word
inanis initium()
{
    scriboln("Alo mundo");

}
```

1.8.2 Série de Fibonacci

```
duma fibonacci
var{
        integer a, b, auxiliar, i, n;
}
inanis initium()
{
  a = 0;
  b = 1;
  i = 0;

  scriboln("Digite um numero: ");
  lectio(n);
  scriboln("Serie de Fibonacci:\n");
  scriboln(b , "\n");

  dum(i<n)
  {
    auxiliar = a + b;
    a = b;
    b = auxiliar;
    i = i + 1;

    scriboln(auxiliar , "\n");
  }
}
```

1.8.3 Shell sort

```
duma shell_sort

var{
        integer n, a, i, j, value, gap;
}

inanis initium(){
    scriboln("Digite a quantidade de numeros a serem ordenados: ");
    lectio(n);
    matrix integer vetor[n];
    scriboln("Digite os numeros:");
        quia a in spatium(1,n,1){
        lectio(vetor[a]);
    }

    gap = 1;
        facite{
                gap = 3*gap+1;
        }dum(gap < n);

        facite{
                gap = gap / 3;
                quia a in spatium(gap,n,1){
                        value = a[i];
                        j = i - gap;

                        dum(j >= 0 && value < a[j]) {
                        a[j + gap] = a[j];
                        j = j - gap;
                        }
                        a[j + gap] = value;
                }
        }dum(gap > 1);
```

```
    scriboln("Numeros ordenados:");
        quia a in spatium(1,n,1){
                scriboln(vetor[a]);
        }
}
```