UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DA COMPUTAÇÃO
ENGENHARIA DA COMPUTAÇÃO

Especificação da linguagem DUMB

Compiladores 2022.1 - COMP379-M

Aluno: Rafael Galhós Marinho Gomes
Professor: Alcino Dall'Igna Junior

Maceió - 2022

1. Nomes

1.1. Identificadores

Um nome válido de identificador deve ter no máximo 31 caracteres e começar com uma letra seguida de zero ou mais letras, dígitos, underscores (_) e/ou cifrões ($). A linguagem diferencia letras maiúsculas e minúsculas.

```
[a-z][a-z0-9_$]{0,30}
```

1.2. Palavras reservadas

A linguagem DUMB reserva as seguintes palavras, não permitindo que o programador declare variáveis ou funções com esses nomes:

int	**uint**	**double**	**char**	**string**
var	**const**	**function**	**return**	**void**
if	**else**	**while**	**for**	**break**
true	**false**	**continue**	**readv**	**printv**
bool				

2. Tipos de dados

Esta seção especifica os tipos de dados primitivos, como as variáveis são inicializadas e coerções para booleano e mais tipos.

2.1. Tipos primitivos

2.1.1. int

- Usado para armazenar números inteiros positivos e negativos;
- Ocupa 4 bytes na memória;
- Inicializado com o valor $\mathbf{0}_{10}$;

2.1.2. uint

- Armazena números inteiros sem sinal;
- Ocupa 4 bytes na memória;
- Inicializado com o valor $\mathbf{0}_{10}$;

2.1.3. double

- Armazena pontos flutuantes de precisão dupla no padrão IEEE 754;
- Inicializado com o valor $\mathbf{0}_{10}$;

2.1.4. char

- Utilizado para armazenar caracteres da tabela ASCII ou inteiros;
- Ocupa 1 byte da memória e não leva em conta o sinal;
- Inicializado com o valor $\mathbf{0_{10}}$;

2.1.5. string

- Na linguagem DUMB, *string*s são consideradas tipos primitivos, mas são representadas com vetores de *char*s internamente;
- O tamanho ocupado por uma string na memória é `(tamanhoCadeiaChars + 1)` bytes;
- Inicializada com uma cadeia de caracteres vazia, `""`.

2.1.6. bool

- Ocupa 1 byte na memória;
- Inicializado com o valor $\mathbf{0_{10}}$.

A linguagem também suporta arranjos unidimensionais (arrays), que são especificados na declaração de variáveis.

2.2. Coerção de tipos

2.2.1. int

- Na conversão para booleano, $\mathbf{0_{10}}$ e $\mathbf{-0_{10}}$ são `false` e qualquer outro valor é `true`;
- A conversão para **double** é automática. O sinal continua o mesmo e a parte decimal é preenchida com zeros;
- Na conversão para **char**, apenas o último byte menos significativo é utilizado.

2.2.2. uint

- Na conversão para booleano, $\mathbf{0_{10}}$ é `false` e qualquer outro valor é `true`;
- A conversão para **double** é automática. O sinal é sempre positivo e a parte decimal é preenchida com zeros;
- Na conversão para **char**, apenas o último byte menos significativo é utilizado.

2.2.3. double

- Na conversão para booleano, $\mathbf{0_{10}}$ e $\mathbf{-0_{10}}$ são `false` e qualquer outro valor é `true`;
- A conversão para **int** é automática. O sinal é mantido e apenas a parte inteira do número é passada para a outra variável;
- Na conversão para **char**, apenas o último byte menos significativo da parte inteira é utilizado.

2.2.4. char

- Na conversão para booleano, $\mathbf{0_{10}}$ é `false` e qualquer outro valor é `true`;
- A conversão para **double** e **int** é automática.

2.2.5. string

- Na conversão para booleano, uma string de tamanho zero é convertida para `false` enquanto qualquer outro caso é `true`.

2.2.6. bool

- Na conversão para int, uint, double e char, assume o valor $\mathbf{0_{10}}$ caso seu valor seja $\mathbf{0_{10}}$ na memória e $\mathbf{1_{10}}$ caso seja qualquer outro valor.

3. Variáveis

3.1. Escopo

O escopo é delimitado pelos caracteres `{` e `}`.
A linguagem DUMB aceita declarações de variáveis e constantes tanto no escopo global quanto dentro do escopo de funções.

3.2. Declaração

A declaração de uma variável deve ser feita da seguinte forma:

```
var nome: tipo; // Variável não inicializada
var nome: tipo = valor; // Variável inicializada
```

Exemplo:

```
var valor: int;
var pi: double = 3.14159265;
var nome: string = "Beethoven";
```

3.3. Vinculação

As variáveis devem ser declaradas explicitamente e seus tipos são estáticos, ou seja, não podem mudar durante a execução do programa.

3.4. Armazenamento e tempo de vida

As variáveis são estáticas e vinculadas em tempo de compilação. Variáveis globais permanecem na memória até que a execução do programa se encerre. Variáveis locais (dentro do escopo de funções) são variáveis dinâmicas da pilha, neste caso a vinculação ocorre em

tempo de execução e elas permanecem na memória apenas enquanto a função está sendo executada.

3.5. Ordem de declaração

Todas as variáveis devem ser declaradas explicitamente antes de serem utilizadas.

3.6. Vetores

Um vetor é um grupo de elementos de mesmo tipo alocados sequencialmente na memória.

3.6.1. Declaração de vetores

A declaração de um vetor deve ser feita da seguinte forma:

```
var nome: tipo[tamanho];
var nome: tipo[tamanho] = [v, a, l, o, r, e, s];
```

3.6.2. Atribuição de valores

Um valor pode ser atribuído a uma posição específica de um da seguinte forma:

```
nome[posição] = valor;
```

4. Constantes

Constantes seguem as mesmas regras que as variáveis, porém o valor de uma constante deve ser sempre especificado na inicialização e não pode ser alterado.

```
const pi: double = 3.14159265;
const valores: int[5] = [1, 2, 3, 4, 5];
```

5. Precedência de operadores

A avaliação dos operadores é feita na seguinte ordem:

```
(   )
+   - (unários)
    !
    ^
/   *   %
+   - (binários)
<   <=   >   >=
```

== !=
&&
||
=

6. Estruturas de controle

6.1. Estrutura de iteração

Os testes são feitos antes de cada iteração.

```
for (atribuição; expressão lógica; atribuição) {
      declaração;
}


while (expressão lógica) {
      declaração;
}
```

6.2. Estrutura condicional

A forma geral é dada por:

```
if (expressão lógica) {
      declaração;
} else if (expressão lógica) {
      declaração;
} else {
      declaração;
}
```

6.3. Desvio incondicional

O comando `break` termina a execução do laço, retomando a execução do programa a partir do próximo comando após o laço.

O comando `continue` força uma nova iteração do laço, antes executando o incremento (no caso do `for`) e verificando novamente a condição de execução.

7. Entrada e saída

7.1. Entrada

O programa lerá valores através do comando `readv`.

```
var x: int;
```

```
var y: int;
var z: int;
readv x, y, z;
```

7.2. Saída

O programa escreverá através do comando `printv`.

```
var x: int = 15;
return "O valor da variavel x e: ", x, "\n";
```

8. Comentários

Comentários são feitos através da sequência de caracteres **`//`**. O analisador ignorará tudo até a próxima quebra de linha.

```
// Comentário!
var numero: int = 73;  // Mais um comentário :D
numero = numero * 5;   // Outro comentário
```

9. Funções

Funções são definidas da seguinte forma:

```
function nome(arg1: tipo1, arg2: tipo2): tipo_retorno { }
```

Exemplo:

```
function media(a: int, b: int): int {
     return (a + b) / 2;
}
```

Caso a função não retorne nenhum valor, seu tipo de retorno deverá ser void.