Universidade Federal de Alagoas

Instituto de Computação
Curso de Ciência da Computação

Linguagem D

Especificação da Linguagem

Arthur Sávio Bernardo de Melo
João Messias Lima Pereira

Maceió 2020.1

Introdução

A linguagem de programação D foi criada com inspiração nas linguagens de programação C e Python, tendo como meta de utilização o ensino instrutivo da programação. D não é orientada a objetos e é case-sensitive - diferencia letras maiúsculas de minúsculas -. Partindo para legibilidade, D não faz coerção (não admite conversões implícitas de tipo) e possui palavras reservadas, que estão em inglês.

Estrutura geral do programa

● Ponto de início de execução

O ponto inicial de execução do programa será identificado por uma construção específica da linguagem, sempre será a função main() com o tipo de retorno obrigatório int.

```
int main() {
        .
        .
        .
        return 0;
}
```

● Definições de procedimentos / funções

Poderão ser declaradas dentro do escopo global e seu acesso só será permitido se tiverem sua implementação declarada antes da função chamadora. A linguagem também não aceita a passagem de subprogramas como parâmetros. A declaração de função é sempre iniciado pelo seu tipo de retorno obrigatório (int, float, bool, string, char), seu identificador único e uma lista de parâmetros (os parâmetros acompanharão de seu tipo e seu identificador) delimitada por abre e fecha parênteses **( )**. A abertura e fechamento do bloco é feita por abre e fecha chaves .

A declaração de um procedimento é feita seguindo o padrão de função, porém o tipo de retorno deve ser identificado como void. A passagem de parâmetro para os procedimentos e funções se dá por meio do modo de entrada e saída, para todos os tipos e estruturas.

Os parâmetros podem ser formais ou efetivos. Os parâmetros formais são usados na declaração da função, por isso denominados formais e os efetivos são usados na chamada da função, que devem ser providenciados por quem invoca a função. Tanto a passagem dos parâmetros como o retorno ocorrem pela passagem de valor ou pela passagem de referência.

Ex.:

```
tipoDoRetorno id ( listaDeParametros ) {
        .
        .
        .
        return x;
}

int somar_numeros (int a, int b) {
        .return a + b;
}
```

A declaração de assinaturas seguirá o padrão de função ou procedimento, com exceção de que finaliza com um ponto e vírgula ao invés de abre e fecha chaves.

Ex.:

```
tipoDoRetorno id ( listaDeParametros );
void id ( listaDeParametros );
```

A chamada de funções seguirá o seguinte padrão: id (listaDeParametros);

Ex:

```
int a = 2, b = 3;
somar_numeros(a, b);

somar_numeros(3, 4);
```

Conjuntos de tipos de dados e nomes

● Identificador

Possuirão sensibilidade à caixa, limite de tamanho de 31 caracteres e seu formato será definido a partir da seguinte expressão regular:

‘[A-Za-z0-9_]+’

Exemplos de nomes aceitos pela linguagem: _var1, var2, var, _, _a;
Exemplos de nomes não aceitos pela linguagem: .abc, ab@c, 1ac.

● Palavras reservadas

As palavras reservadas são sempre escritas em inglês e são listadas a seguir:

bool, char, const, else, false, float, for, if, int, len, print, return, scan, string, true, void, while.

● Definição de variáveis

D possui escopo global e local, logo as variáveis podem ser definidas fora do escopo de qualquer função ou dentro das mesmas. As variáveis declaradas no escopo global podem ser acessadas globalmente ou dentro do escopo da função como variáveis locais. Já as variáveis declaradas dentro do escopo de uma função só poderão ser acessadas dentro da função em que foi declarada. Para declarar uma variável inicia-se com o **tipoDaVariável(int, float, bool, string e char)** e seguidas pelo seu identificador que é único.

Múltiplas variáveis podem ser declaradas em uma declaração de tipo único sendo separadas por vírgula. Cada declaração termina com um ponto e vírgula e podem ser inicializadas. A inicialização das variáveis ocorre com a declaração da variável seguido do símbolo de atribuição seguido da expressão.

Para declaração da variável
Ex.:

```
int a;
float b;
bool c;
char d;
string e;
```

Para inicialização da variável

Ex.:

```
int a = 1;
float b = 1.5;
bool c = true;
char d = ‘j’
string e = “hello”
```

● Definição de constantes

Constantes possuem o mesmo padrão de declaração e definição de variáveis, com exceção que iniciam sempre com a palavra reservada const e que a atribuição de valor é obrigatória na declaração.

Ex.:

```
const int teste = 0;
```

Tipos e Estrutura de Dados

● Inteiro

É a identificação da variável como sendo do tipo inteiro de 32 bits, identificado pela palavra reservada int. Seus literais são expressos como uma sequência de dígitos decimais.

Ex.:

```
int a  = 1;
int b = 1234;
int c = 123456;
```

● Ponto Flutuante

É a identificação da variável como sendo do tipo ponto flutuante de 64 bits, identificado pela palavra reservada float. Seus literais são expressos como uma sequência de dígitos decimais, seguido de um ponto e os demais decimais.

Ex.:

```
float a = 1.5;
float b = 123.45
flota c = 11.234
```

● Caractere

É a identificação da variável como sendo do tipo caractere de 8 bits, identificado pela palavra reservada char. Guarda um código referente ao seu símbolo na tabela ASCII. A constante literal do caractere é delimitada por apóstrofo.

Ex.:

```
char c = ‘a’;
```

● Cadeia de Caracteres

É a identificação da variável como sendo do tipo cadeia de caracteres, identificado pela palavra reservada string. Seus literais são expressos como um conjunto de caracteres, mínimo de 0 caracteres e de tamanho máximo ilimitado. A constante literal é delimitada por aspas.

Ex.:

```
string str;
string s = “teste”;
```

● Booleano

É a identificação da variável como sendo do tipo booleana, identificado pela palavra reservada bool. Possui dois valores possíveis: true, false.
Ex.:

```
bool a;
bool b = true;
```

● Ponteiros

Ponteiros podem ser de qualquer tipo, e armazenam um endereço de memória que pode ser acessado e manipulado.
Ex.:

```
float* b;
char *c;

int a;
int *ba = &a;
```

● Arranjos unidimensionais

Os arranjos unidimensionais são compostos pelos tipos determinados acima. Sua declaração iniciará com o tipo seguido do identificador único e, delimitado por abre e fecha colchetes [ ], o tamanho do arranjo. Há um comando da linguagem que retorna um inteiro indicando o tamanho do arranjo. O comando é representado pela palavra reservada **len** seguida pela identificação do arranjo informada dentro de parênteses. O comando len retorna o tamanho do arranjo em formato inteiro.
.
Exemplo dos arranjos.:

```
    int arr[9];
    float arr[1024];
    bool arr[] = {true, false, false, true};
```

Exempo de len:

```
    int s = len(arr);
```

Conjunto de Operadores

● Aritméticos

- (unário negativo)
* (multiplicação)
/ (divisão)
% (resto)
+ (soma)

- (subtração)

● Relacionais

< (menor que)
<= (menor ou igual que)
> (maior que)
>= (maior ou igual que)
== (igual)
!= (diferente)

● Lógicos

! (negação unária)
&& (conjunção)
|| (disjunção)

● Referenciais

O endereço de memória de uma variável é dado pelo operador & e pode ser aplicado a todos os tipos aceitos pela linguagem. Assim, se i é uma variável então &i é o seu endereço.

Um ponteiro é um tipo especial de variável que armazena um endereço. A utilização ocorre pela inserção do operador '*' (ponteiro) antes do identificador da variável.

Ex.: O programa abaixo exemplifica a utilização dos operadores. 'b', como é um ponteiro, recebe o endereço de memória da variável a, através do operador &.

```
int main() {
        int a = 5;
        int *b;

        b = &a;
}
```

● Concatenação de cadeia de caracteres

O operador de concatenação é dado por **::**, pode ser aplicada a todos os tipos aceitos pela linguagem (int, float, bool, char, string). Os tipos numéricos ou booleanos, assim como caracteres e strings, quando concatenados, geram sempre uma nova cadeia de caracteres.

● Coerção

D não aceita coerção entre variáveis de tipos diferentes. Dessa forma, todas as verificações de compatibilidade de tipo serão feitas estaticamente.

● **Operações suportadas**

Operador	Tipos que aceitam a operação
‘!’	bool
‘ + ’, ‘ - ’, ‘ * ’, ‘ / ’	int, float
‘ < ’, ‘ > ’, ‘ <= ’, ‘ >= ’	int, float, char
‘==’, ‘ != ’, ‘ = ’, ‘ :: ’	int, float, bool, char, string
‘&’	int, float, bool, char, string
*	int, float, bool, char, string
‘&&’, ‘||’	bool

● **Precedência e Associatividade**

Operadores (Precedência)	Associatividade
‘&’ endereço de variável e ‘*’ ponteiro	Não Associativo
‘!’ negação	Direita para esquerda
‘-’ unário negativo	Direita para esquerda
‘*’ multiplicação e ‘/’ divisão	Esquerda para direita
‘+’ adição e ‘-’ subtração	Esquerda para direita
‘<’ menor que, ‘<=’ menor ou igual que, ‘>’ maior que, ‘>=’ maior ou ou igual que	Não associativo
‘==’ igualdade e ‘!=’ diferente	Não associativo
‘&&’ lógico e	Esquerda para direita
‘||’ lógico ou	Esquerda para direita
‘::’ concatenação	Esquerda para direita
‘=’ atribuição	Direita para esquerda

Constantes literais e suas Expressões

As expressões regulares das constantes literais são denotadas da seguinte maneira:
1. Constantes de inteiros: ‘[:digit:]+’

2. Constantes de floats: ‘[:digit:]+\.[:digit:]+’

3. Constantes de char: '\’[\x00-\x7F]\’’

4. Constantes de bool: “true"| "false”

5. Constantes de string: ‘\"[\x00-\x7F]*\"’

Obs: digit é o padrão Flex para os dígitos de 0-9.

.

Comandos

● Atribuição

É definida pelo comando ‘=’, sendo o lado esquerdo o identificador único e o lado direito o valor ou expressão. Pode ser utilizada para atribuir valores de quaisquer tipos aceitos pela linguagem a um identificador.

● Estrutura condicional

Uma estrutura condicional controla se o próximo bloco de sentenças será executado ou não. O bloco de sentenças é delimitado por chaves **{ }**. O controle da execução do bloco de sentenças se dá pela análise de uma expressão lógica informada dentro de parênteses **( )**. Se a expressão resultar em true, então as sentenças são executadas. Se a análise da expressão resultar em false, todo o bloco de sentenças é ignorado. Para definir uma estrutura condicional pode-se utilizar as palavras reservadas **if** e **else**.

O primeiro bloco de sentenças deve sempre começar com um **if**. A linguagem permite a utilização de **ifs** aninhados. Por fim, a palavra reservada **else** é associada a um if anterior e será executada caso a expressão lógica do if retorne falso. Esse bloco é opcional e pode ser omitido

Ex:

```
if ( expressaoLogica ) {
        .
        bloco de sentenças
        .
} else {
        .
```

bloco de sentenças

.

}

● Estrutura interativa com controle lógico

Uma estrutura interativa com controle lógico é uma estrutura de repetição que executa um bloco de sentenças enquanto uma determinada condição é verdadeira. A expressão lógica será sempre analisada antes da execução de uma repetição. Caso continue sendo verdadeira, o bloco de sentenças será executado novamente. Do contrário, o bloco de sentenças é ignorado. A linguagem também permite que estruturas interativas possam ser aninhadas. Para declarar uma estrutura interativa pode-se utilizar a palavras **while**.

A palavra reservada **while** deve ser seguida de uma expressão lógica dentro de parênteses ( ) e de um bloco de sentenças dentro de chaves { }.

Ex.:

while (expressaoLogica) {

.

bloco de sentenças

.

}

● Estrutura interativa com controle por contador

Para utilizar a estrutura de controle por contador, deve-se utilizar a palavra reservada **for**. Ela deve ser seguida por um cabeçalho, informado dentro de parênteses ( ). O cabeçalho possui uma declaração de variável que será utilizada para o controle do laço e seguida de três parâmetros: inicialização, valor final e passo. A inicialização corresponde a um inteiro que será atribuído à variável declarada. Em seguida, o valor final corresponde ao número limite que a variável pode ter, sendo esse valor a condição de parada a ser analisada. Por fim, o passo corresponde ao número de incremento do valor da variável a cada repetição do laço. Qualquer um dos três parâmetros poderá ser preenchido com uma outra variável, desde que seja do tipo inteiro, para servir como os valores do cabeçalho.

A inicialização ocorre antes da primeira repetição. Antes de cada repetição ser executada, o programa avalia o valor final e só assim executa o bloco de sentenças caso o valor da variável seja menor que o valor final. Ao final de cada repetição a variável contadora será atualizada com o valor informado no cabeçalho. Seguido do cabeçalho, deve-se informar um bloco de sentenças dentro de chaves { }.

Ex.:

for (declaração var: (valor inicial, valor final, passo)) {

.

bloco de sentenças

.

```
}

for (int i:(0, 10,  1)) {
        .
        bloco de sentenças
        .
}

for (int j:(3, 1000,  3)) {
        .
        bloco de sentenças
        .
}
```

Desvios Incondicionais

● **Return**

A linguagem permite apenas um único desvio incondicional. Return é utilizado quando deseja-se que o controle retorna para a origem. A palavra reservada para este comando é **return** tipo aceito pela linguagem.

Entrada e Saída

● **Entrada:** É realizada através do comando **scan()**.

O comportamento da função scan dependerá da variável a qual está sendo lida. No caso de ser um inteiro, scan atribui um único inteiro ao valor designado ao identificador, assim como floats, char, string, bool. A função scan também aceita vários parâmetros, atribuindo, assim, cada um dos valores recebidos como entrada às respectivas variáveis.

Ex.: *scan(variavelString, variavelInteira, variavelFloat, variavelCaractere, variavelBool)*

● **Saída:** É realizada através do comando **print()**.

O comportamento da função print dependerá do tipo da variável que será passada como parâmetro. No caso de ser um inteiro, print irá mostrar na tela um único inteiro: o valor designado ao identificador, assim como para floats, char, string, bool. Caso o valor seja relacionado a uma constante, print irá imprimir diretamente o valor dessa constante. Se for um arranjo, print informará os elementos do arranjo, separados por vírgula.

A saída poderá ser formatada, mas isso é opcional. Para cada tipo há um identificador. %d para inteiro, %f para float, %c para caractere, %s para string e %b para bool. Para ponto flutuante poderá ser especificado o número de casas decimais informado com um ponto e seguido pelo número de casas a serem apresentadas (%f.inteiro literal), por padrão serão apresentadas duas casas decimais.

Ex:

Saída formatada:

print(“%f.3”, variavelFloat);

print(“%s, %d, %f, %c, %b”, variavelString, variavelInteira, variavelFloat, variavelCaractere, variavelBooll)

Saída padrão:

print(identificadorVariavel)

Códigos exemplo

- **Hello world**

```
int main() {
        print("Hello World!");
}
```

- **Fibonacci**

```
void fib(int n)
{
        int fib1 = 1, fib2 = 1, soma;

        while(fib1 < n) {
                print(fib1);

                soma = fib1 + fib2;
                fib1 = fib2;
                fib2 = soma;
        }
}

int main()
{
        int n;
        scan(n);
        fib(n);

        return 0;
}
```

- **Shell Sort**

```
void shellSort(int *vet)
{
   int i, j, value;
```

```
    int size = len(vet);
    int h = 1;

    while (h < size) {
        h = 3 * h + 1;
    }

    while (h > 0) {
        for(i : (h, size, 1) {
            value = vet[i];
            j = i;
            while (j > h-1 && value <= vet[j - h]) {
                vet[j] = vet[j - h];
                j = j - h;
            }
            vet[j] = value;
        }
        h = h/3;
    }
}

int main()
{
    int size;
    scan(size)

    int array[size];

    for(int i : (0, size, 1)) {
        scan(array[i])
    }

    for(int i : (0, size, 1)) {
        print(array[i])
    }

    shellSort(&array);

    for(int i : (0, size, 1)) {
        print("%d", array[i])
    }

    return 0;
}
```

Referências

- https://www.ic.unicamp.br/~wainer/cursos/2s2011/Cap05-EstruturasCondicionais-texto.pdf
- https://www.ic.unicamp.br/~wainer/cursos/2s2011/Cap06-RepeticaoControle-slides.pdf
- https://www.ime.usp.br/~leo/mac2166/2017-1/introducao_parametros_funcoes.html