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, que é mandado uma cópia do valor da variável.

Ex.:

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

void id ( listaDeParametros ) {
        .
        .
        .
}
```

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 );
```

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:

‘_{0,1}[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:

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

● 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 e acessadas globalmente ou dentro do escopo da função como variáveis locais. 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 acompanhadas de uma atribuição. A inicialização das variáveis ocorre do mesmo modo da atribuição, com a declaração da variável seguido do símbolo de atribuição seguido à 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 s;

● Booleano

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

● Ponteiros

Ponteiros podem ser de qualquer tipo, e armazenam uma referência a memória que pode ser acessada e manipulada.
Ex.: int *a;
float* b;
char *c;

● 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);
```

● 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

● Coerção

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

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 operador de referência é dado por &, pode ser aplicado a todos os tipos aceitos pela linguagem. Gerando como resultado um ponteiro que aponta para o endereço de memória que guarda o valor da variável onde foi utilizado o operador.

O operador de desreferência é dada por *, usado unicamente do lado esquerdo de um identificador, seu uso permite acessar o valor de memória referenciada por um ponteiro. Ex.: *a = 2

● 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.

● Precedência e Associatividade

Operadores (Precedência)	Associatividade
‘&’ e ‘*’	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 um valor diferente de true, então as sentenças são executadas. Se a análise da expressão resultar em false, todo o bloco de sentenças é ignorado e o programa parte para o próximo bloco, se houver, ou continua a leitura do restante do código. Para definir uma estrutura condicional pode-se utilizar as palavras reservadas: **if**, **elif** e **else**.

O primeiro bloco de sentenças deve sempre começar com um **if**. Podem ser adicionados quantos **if 's** forem necessários, sem que um dependa da condição do outro. Para casos em que mais de uma condição precise ser analisada e após isso executar um bloco de sentenças diferente do anterior, utiliza-se **elif** para designar a próxima estrutura condicional. A estrutura elif pode ser utilizada quantas vezes forem necessárias para descrever a lógica de um algoritmo.

Por fim, a palavra reservada else é associada às estruturas condicionais anteriores e será executada caso todas as estruturas condicionais não forem verdadeiras. Esse bloco é opcional e pode ser omitido

Ex:

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

A palavra reservada **while** está associada com a estrutura de interação por controle lógico e 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 iterativa controlada por contador

Essa estrutura executa um bloco de sentenças controlada por um contador que é atualizado a cada repetição. Dentro do cabeçalho estão as informações de inicialização, condição e atualização. A inicialização ocorre antes da primeira repetição. Antes de cada repetição ser executada, o programa avalia a condição e só assim executa o bloco de sentenças caso a condição seja verdadeira. Ao final de cada repetição o contador será atualizado.

Para utilizar uma estrutura interativa controlada por contador utiliza-se a palavra reservada **for**, seguido das instruções de inicialização, condição e atualização dentro de parênteses **( )** e separados por ponto e vírgula. O bloco de sentenças deve ser informado dentro de um de chaves **{ }**.

Ex.:

```
for (int i: (inicialização, i operadorLógico, identificador ou número, atualização)) {
        .
        bloco de sentenças
        .
}

for(int i:(0, i < 10,  1)) {                    -> Este bloco será executado 10 vezes
        .
        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** e deve ser seguida de um inteiro.

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 para o código atribuído às respectivas variáveis. Cada parâmetro a ser lido precisa de um identificador. Para cada tipo aceito na linguagem há um identificador. %d para inteiro, %f para float, %c para caracteres, %s para string e %b para bool.

Ex.: scan(“%s, %d, %f, %c, %b, &string, &inteiro, &caractere, &bool)

● **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. Para cada tipo há um identificador. %d para inteiro, %f para float, %c para caractere, %s para string e %b para bool.

Ex: print(“%s, %d, %f, %c, %b”, &string, &inteiro, &float, &caractere, &bool)

Códigos exemplo

- **Hello world**

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

- **Fibonacci**

```
void fib(int n)
{
```

```
        int fib1 = 1, fib2 = 1, soma;

        for (int i : (3, n, 1))
        {
                print("%d", fib1);

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

int main()
{
        int n;
        scan("%d", &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 len;
   scan("%d", &len)

   int array[len];

   for(int i : (0, len, 1)) {
      scan("%d", &array[i])
   }

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

   shellSort(vet);

   for(int i : (0, len, 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