

Universidade Federal da Fronteira Sul - UFFS

Linguagem de programação Lotus

Acácia dos Campos da Terra
Gabriel Batista Galli
Vladimir Belinski

Chapecó
2015

A linguagem de programação Lotus é uma linguagem desenvolvida com base em Java com as seguintes características e funcionalidades:

- Extensão de arquivos
- Sintaxe
- Declaração de variáveis
- Atribuições de valor
- Conversão de tipos
- Operações aritméticas
- Entrada e saída
- Desvios condicionais
- Laços de repetição

Extensão de arquivos

Os códigos escritos com a linguagem de programação Lotus devem apresentar extensão “.lt”.

Sintaxe

A sintaxe da linguagem Lotus de programação é flexível, ou seja, os comandos podem apresentar quebras de linha que podem incluir até mesmo comentários, caso seja conveniente.

Já comandos do tipo if, for e while exigirão ‘{’ e ‘}’ como delimitadores de bloco. Comentários são marcados com “--”, assim como o “//” do C ou Java. Não há comentários de bloco.

Declaração de variáveis

A sintaxe de declaração de variável funciona da seguinte forma:

```
let nome_variavel: tipo_variavel;
```

Sendo obrigatório o uso de “let ” antes de qualquer declaração e o uso de ‘;’ ao final dela. É permitida a criação de várias variáveis do mesmo tipo em apenas uma linha, desde que separadas por vírgulas, exemplo:

```
let x, y, z: int;
```

Também é permitida a atribuição de valores às variáveis no momento da declaração. Exemplo:

```
let x = “Oi”: string;
```

Ao declarar variáveis, é importante saber:

Variáveis não podem iniciar com números e o único carácter permitido em uma variável é ‘_’. Exemplo:

```
let _oi_var: string;
```

```
let oi88: double;
```

São exemplos de uso errôneo ao declarar variável:

```
let 9var: int; --(não se inicia com números)
```

```
let var@: int; --(não é permitido o uso de caracteres especiais)
```

Atribuições de valor

Para atribuir valores à uma variável, basta fazer:

```
nome_variavel = valor_variavel;
```

As atribuições também podem ser feitas na própria declaração de variável, como demonstrado.

Se por ventura for atribuído um valor diferente do do tipo da variável, então será automaticamente convertido. Exemplo:

```
let b: bool;
```

```
b = 861;
```

b após a conversão automática valerá “true”.

Conversão de tipos

Todas as variáveis que receberem um valor que não seja compatível com o seu próprio tipo (especificado na declaração) passarão por uma conversão automática, onde:

Se foi atribuído um número inteiro:

- à uma variável double: se o int era x, o double será x.0;
- à uma variável bool: se o int era 0, torna-se false, senão true;
- à uma variável string: se torna uma string que representa o valor inteiro;

Se for atribuído um número double:

- à uma variável int: o valor será truncado, ou seja, só será guardado o que está antes da vírgula;
- à uma variável bool: se o valor for 0.0, torna-se false, senão true;
- à uma variável string: torna-se uma string que representa o valor double;

Se for atribuído um valor bool:

- à uma variável int: se for true, torna-se 1 e se false, torna-se 0;
- à uma variável double: se for true, torna-se 1.0, se false, 0.0;
- à uma variável string: torna-se uma string de conteúdo "true" ou "false", dependendo do valor da variável;

Se for atribuído um valor de uma string:

- à uma variável int ou double: a variável receberá o comprimento equivalente da string;
- à uma variável bool: será true se a string é exatamente "true". Caso contrário, false;

Operações aritméticas

Todos os tipos de variáveis operam entre si, através da conversão de tipos, utilizando os seguintes operadores:

Matemáticos: -, +, /, %, *, ^;

Comparação: <, <=, ==, >=, >, !=;

Booleanos: !, &&, ||;

Exemplo de uso:

```
let x: int;
```

```
let y: double;
```

```
x = 3 - -4;
```

```
y = + 2 ^ - 4 + -0 / + 5;
```

Onde a resposta será $x = 7$ e $y = 0.0625$.

Os operadores de comparação e os booleanos sempre resultam em um booleano. A comparação de strings é em relação a ordem lexicográfica (se as strings têm tamanhos diferentes, a menor vem antes). Comparações com booleanos são feitas de modo que o valor é primeiro transformado em inteiro e só então é feita a comparação (exceto para os operadores “==” e “!=”).

Os operadores matemáticos variam com seu tipo:
Int e double funcionam normalmente, como ensinado na matemática.

Para booleanos, os operadores matemáticos são interpretados como as seguintes operações lógicas:

+ → or;
- → nor;
/ → nand;
% → xnor;
* → and;
^ → xor;

Para string, em relação aos operandos:

+ → concatena o segundo ao primeiro;
- → retira a primeira ocorrência do segundo no primeiro;
/ → retira todas as ocorrências do segundo no primeiro;
% → retira todas as ocorrências do primeiro no segundo (“resto” da operação de ‘/’);
* → adiciona o segundo entre cada caractere do primeiro;
^ → se o segundo operando for uma string, retornará o índice daquela string dentro do primeiro operando. Caso contrário, retorna o char do primeiro na posição indicada pelo segundo;

Entrada e saída

Para imprimir uma linha, deve ser chamada a função print(). A mesma é equivalente ao printf() do C ou System.out.print() do java. O comando println() imprimirá o que foi solicitado e também imprimirá uma quebra de linha ao final (equivalente ao System.out.println() do Java).

Para imprimir o valor de uma variável, a mesma deve ser expressa entre cifrões. Exemplo:

```
let ex: string;
```

```
print(var: $ex$);
```

Onde “print” é o comando e o uso de ‘(’ e ‘)’ é obrigatório para delimitar o comando. “var: ” é uma string a ser posta diretamente na tela e \$ex\$ é o nome da variável, que está delimitada pelos cifrões para poder ser impressa.

Os caracteres especiais em um comando de saída são:

\t para uma tabulação;

\n para uma quebra de linha (para imprimir o literal “\n”, faça “\\n”);

\\$ para um cifrão;

\\ para uma contra-barra;

\- para um hífen

Os comandos “print” e “println” aceitam também expressões em sua forma pura, desde que delimitadas pelos cifrões, como se fosse uma variável e então as expressões são resolvidas. Exemplo:

```
println($ -3 + 4 * 2 / ( 1 - 5)^ 2 ^ 3$);
```

O qual a resposta será -3.

De maneira análoga às funções de saída, o comando scan() é utilizado para ler um valor do teclado. Esse comando lê a informação até o primeiro espaço ou quebra de linha que delimitar a entrada desejada, sendo idêntico ao scanf() do C. Já o comando scanln() lê uma linha inteira, assim como o fgets() do C. Em ambos os comandos, a conversão dos tipos de variável se aplica, caso a entrada não corresponda ao valor esperado.

Exemplo do comando scan():

```
let i: int;
```

```
scan(i);
```

Onde “scan” é o comando, ‘(’ e ‘)’ são os caracteres obrigatórios de delimitação e “i” é a variável onde será armazenado o valor lido. Neste exemplo, um valor do tipo inteiro.

Desvios condicionais

A linguagem suporta toda e qualquer cadeia ou aninhamento de comandos `if`. A condição é qualquer expressão suportada pela linguagem, cujo resultado será transformado em booleano para efeitos de avaliação.

Um comando de desvio condicional se comportará da seguinte forma:

```
if (condição) {  
    -- comandos a serem executados quando a condição for verdadeira  
}  
elsif (outra condição) {  
    -- comandos a serem executados quando a segunda condição for verdadeira  
}  
else {  
    -- comandos a serem executados quando nenhuma das condições acima for verdadeira  
}
```

Sendo “`if`”, “`elsif`” e “`else`” palavras reservadas para as operações de desvio e o uso de ‘(’ e ‘)’ obrigatórios para delimitar a(s) condição(ões). O uso de ‘{’ e ‘}’ também é obrigatório para delimitar o bloco de código, mesmo que seja escrito em uma linha.

Laços de repetição

O laço de nome `While` funciona da seguinte forma:

É estabelecida uma variável determinando um início e uma determinando um final, então uma variável irá iterar até atingir esse valor final, executando novamente todos os comandos que estão no bloco do `While`. Exemplo:

```
let x = 0, y = 0: int;
```

```
let s = "Lotus": string;
```

```
y = s; -- Aqui ocorre conversão de tipo, onde y receberá 5, que é o compri-
```


mento de s.

```
while(x < s){  
    println($s ^ x$);  
    x = x + 1;  
}
```