Universidade Federal de Alagoas
Instituto de Computação
Ciência da Computação

Ultima

BELO, Bruno da Silva ROCHA, Wesley Marques

27 de julho de 2016

Sumário

1 Introdução

Ultima se trata de uma linguagem procedural, estaticamente e fortemente tipada. O programa normalmente consiste de um entry-point, a função main, onde o algoritmo será escrito, delimitado por um escopo e cada instrução deverá terminar com um ponto e vírgula.

Será possível a criação de funções para melhorar a legibilidade e escritabilidade do código. Estas funções devem ser declaradas e definidas, não somente declarar, antes do entry-point do programa.

Seja na função main ou uma função qualquer criada, o algoritmo será definido usando declaração e variável, estruturas condicionais e de controle.

2 Conjunto de Tipo de Dados

2.1 ID

Um identificador aceito por Ultima é dado da seguinte forma:

- Inicia-se, obrigatoriamente, com uma letra, maiúscula e minúscula.
- Os demais caracteres podem ser letras, números ou underscore.
- Podem possuir qualquer tamanho, porém todo o nome não pode ter uma quebra de linha no meio.

Exemplo:

```
1  int esta_variavel_pode_ser_declarada; # Certo
2  int esta_variavel_nao
3  _pode_ser_declarada; # Errado
```

2.2 Comentário

O caractere para expressar um comentário é o "#", assim quando este caractere for encontrado, o restante da linha depois dele é ignorada. Ultima não possui comentário de bloco, somente de linha.

2.3 Inteiro

O tipo inteiro é declarado pela palavra reservada "int" seguido do ID da variável. Por ser uma linguagem fortemente tipada, não há coerção entre tipos.

Ela expressa um número inteiro e seus literais são declarado por qualquer número. As operações para este tipo estão em 3.2.

Exemplo:

```
1   int var;
2   var = 1;
```

2.4 Ponto flutuante

O tipo que representa ponto flutuante é declarado utilizando a palavra reservada "float" seguido do ID da variável.

Um literal do tipo ponto flutuante é dado por qualquer número separado por somente um ".", assim antes do "." indica o parte inteira e depois dele é a parte decimal. As operações para este tipo estão em 3.2.

Exemplo:

```
1   float var;
2   var = 1.2;
```

2.5 Caracteres e cadeias de caracteres

Ultima tratará cadeias de caracteres usando apenas o tipo "string", do mesmo modo será tratado caracteres isolados. As operações para este tipo estão em 3.5.

Exemplo:

```
1   string str;
2   str = ''Hello World!'';
3
4   string character;
5   character = ''A'';
```

Dentro de literais de cadeias de caracteres (Delimitados por aspas duplas), palavras reservadas podem ser usados. Quebra de linha dentro de um literal de cadeias de caracteres é proibido.

Exemplo:

```
1   string str;
2   str = ''Esta string nao
3   e valida!'';
```

2.6 Booleano

O tipo booleano é declarado usando a palavra reservada "bool" seguido do ID da variável, os únicos dois possíveis valores para a variável são "true" e "false". As operações para este tipo estão em 3.3 e 3.4.

Exemplo:

```
1   bool var;
2   var = true;
```

2.7 Arranjos unidimensionais

Ultima tratará arranjos unidimensionais como listas com acesso aleatório. Para serem declaradas, usasse a palavra reservada "vector" seguida do tipo que será armazenado na lista, seu ID e um ":" separando o tamanho da lista utilizando um literal ou variável do tipo inteiro.

Exemplo:

```
1   vector int list:3;
2
3   setValueInt(list, index, value);
4   getValueInt(list, index);
5   removeInt(list, index);
6   addInt(list, value); # adiciona no final da lista.
```

Há quatro operações nativas para lista. Estas são getValue, setValue, remove e add cujo final varia a depender do tipo da lista, Int para inteiro, String para conjunto de caractere, Float para ponto flutuante e Bool para booleano. Com elas será possível fazer as manipulações das listas.

3 Conjuntos de Operadores

3.1 Atribuição

A atribuição é feita pelo operador "=", onde do lado esquerdo é o endereço na memória que representa o ID da variável e do lado direito é o valor a ser guardado por este endereço de memória. Os dois lados devem possuir o mesmo tipo, pois a linguagem não permite coerção. Exemplo:

```
1   variable = 10;
```

Porém não é possível declarar ou atribuir um conjunto de variáveis.

Exemplo:

```
1   int a = 1, b = 2, c = 3; # Errado
2   int a = 1;
3   int b = 2;
4   int c = 3; # Certo
```

3.2 Aritméticos

- "+": Operador binário que retorna a soma dos dois operandos.
- "-": Operador binário que retorna a diferença dos dois operandos.
- "*": Operador binário que retorna a multiplicação dos dois operandos.
- "/": Operador binário que retorna a divisão dos dois operandos.
- "%": Operador binário que retorna o resto da divisão dos dois operandos.

O operador unário negativo é "∼". Este irá negar valores aritméticos dos números, tanto inteiro, quanto ponto flutuante.

Exemplo:

```
1  int a;
2  int b;
3  a = a + b;
4  b = b - a;
5  a = (a * b) / 2;
```

3.3 Relacionais

- "==": Operador binário que verifica a igualdade entre dois operandos.
- "!=": Operador binário que verifica a desigualdade entre dois operandos.
- "<": Operador binário que verifica se o operando da esquerda é menor que o operando da direita.
- ">": Operador binário que verifica se o operando da esquerda é maior que o operando da direita.
- ">=": Operador binário que verifica se o operando da esquerda é menor ou igual ao operando da direita.
- ">=": Operador binário que verifica se o operando da esquerda é maior ou igual ao operando da direita.

Exemplo:

```
1  if (a > b) {
2    # do something
3  }
```

3.4 Lógicos

- ¬: Operador unário que nega uma expressão lógica.
- "&": Operador binário que executa um "and" lógico.
- "|": Operador binário que executa um "or" lógico.

Exemplo:

```
1  if (a > b | c == d) {
2    # do something
3  }
```

3.5 Concatenação de cadeias de caracteres

A concatenação de cadeias de caracteres será dada pelo operador binário "+". Este por sua vez, criará uma nova string e atribuirá a ela uma união das duas outros strings.

Exemplo:

```
1  string str = ``This str'' + ``More str'';
```

4 Precedência e Associatividade

Na Tabela 1 mostrará a precedência e associatividade dos operadores, onde a precedência é maior para os que se encontram mais acima.

Operadores	Associatividade à
()	Não associativo
¬	Direita
∼	Não associativo
* / %	Esquerda
+ -	Esquerda
<<= >>=	Não associativo
== !=	Não associativo
& |	Esquerda
=	Direita

Tabela 1: Tabela de precedência e associatividade

5 Instruções

Os conjuntos dos statements é similar ao C e cada statement é terminado por um ";", exceto por if, else, while e for.

5.1 Estrutura condicional de uma e duas vias

if, if-else

A estrutura condicional "if" virá seguida de uma condição lógica dentre de parenteses e seu escopo é definido por chaves. O algoritmo irá executar o código contido no "if", se e somente se, a sua condição lógica for verdadeira, caso essa condição não seja satisfeita, é possível criar, opcionalmente, uma clausula "else", que será executada, se e somente se, a condição do "if" em conjunto for falsa. Exemplo:

```
1  int a = 10;
2  int b = a;
3  if (a == b) {
4    # do something
5  } else {
6    # do anotherthing
7  }
```

5.2 Estrutura iterativa com controle lógico

while

"while" é usado como uma repetição condicional, isto é, a repetição só irá parar se a sua condição for falsa.

Exemplo:

```
1  int a = 0;
2  while (a < 10) {
3    # do something
4    a = a + 1;
5  }
```

5.3 Estrutura iterativa controlada por contador

for

O "for" será uma estrutura de repetição que receberá três parâmetros: índice, limite e passo. Ele irá repetir o bloco de código desejado no intervalo [*índice*, *limite*) variando em *passo* até todo intervalo ser passado.

Exemplo:

```
1  for (int i = 0; 10; 1) { # i ira variar de 0 a 9 ao passo de 1
2    # do something
3  }
```

5.4 Entrada e saída

Ultima terá seis funções nativas responsáveis por entrada e saída, essas seriam:

Para entrada:

- inputString(string str)
- inputInt(int i)
- inputFloat(float f)

Para saída:

- outputString(string str)
- outputInt(int i)
- outputFloat(float f)

5.5 Funções

Ultima não terá suporte para sobrecarga de funções. Para declarar uma função define-se o seu tipo, nome da função e parâmetros com seus tipos dentro de parênteses. Para chamá-la, basta colocar o nome da função e passar seus parâmetros e o retorno é feito pela palavra reservada "return".

Exemplo:

```
1   int sumInt(int x, int y) {
2     return x + y;
3   }
4
5   float sumFloat(float x, float y) {
6     return x + y;
7   }
8
9   void sort(vector int list, int size) {
10    # do shell sort
11  }
12
13  vector int apply_exp(vector int list, int size) {
14    # do median
15  }
16
17  int a;
18  float b;
19
```

```
20    a = sumInt(5, 2);
21    b = sumFloat(2.4, ~6.3);
```

Para Arranjos unidimensionais, a passagem de parâmetros é feita por referência, os outros tipos são por valor.

Para funções há um tipo especial chamado "void", em que significa que tal função retornará nenhum tipo. Não é possível uma variável ser do tipo "void".

Já para conjunto unidimensionais, é necessário colocar a palavra "vector" para indicar que uma lista será passada como parâmetro ou retornada.

6 Exemplos de Códigos

6.1 Olá mundo

```
1   int main() {
2     outputString(''Hello World!'');
3
4     return 0;
5   }
```

6.2 Fibonacci

```
1   int fibonacci(int n) {
2     int f1 = 0;
3     int f2 = 1;
4     int fi = 0;
5
6     if (n < 0) {
7       return 0;
8     }
9
10    outputInt(0);
11    outputString('', '');
12    outputInt(1);
13
14    if (n == 0 | n == 1) {
15      return 1;
16    }
17
18    while(fi < n) {
19      fi = f1 + f2;
20      f1 = f2;
```

```
21        f2 = fi;
22        outputString(``, ``);
23        outputInt(fi);
24      }
25
26      return fi;
27    }
28
29    int main() {
30      int n;
31      inputInt(n);
32
33      int fib = fibonacci(n);
34      # do something with fib
35      return 0;
36    }
```

6.3 Shell sort

```
1     void shellsort(vector int vet, int size) {
2       int value;
3       int gap = 1;
4       while(gap < size) {
5         gap = 3 * gap + 1;
6       }
7
8       while(gap > 1) {
9         gap = gap / 3;
10        for (int i = gap; i < size; i = i + 1) {
11          value = getValue(vet, i);
12          int j = i - gap;
13
14          while(j >= 0 & value < getValue(vet, j)) {
15            setValue(vet, j + gap, getValue(vet, j));
16            j = j - gap;
17          }
18
19          setValue(vet, j + gap, value);
20        }
21      }
22    }
23
24    int main() {
```

```
25      int size;
26      inputInt(size);
27
28      int vet:size;
29      for (int i = 0; size; 1) {
30        int x;
31        inputInt(x);
32        add(vet, x);
33      }
34
35      shellsort(vet, size);
36
37      return 0;
38    }
```