Lucas Peixoto de Almeida Cavalcante

Especificação da linguagem de programação LisC

Brasil

Maceió/Al

Julho de 2019

Lucas Peixoto de Almeida Cavalcante

Especificação da linguagem de programação LisC

Especificação da linguagem de programação LisC, definida pelo aluno, para a disciplina de Compiladores, correspondente à parte da avaliação da AB1 do semestre de 2019.2, sob orientação do **Prof. Alcino Dall Igna Jr.**

Universidade Federal de Alagoas

Instituto de Computação

Brasil

Maceió/Al
Julho de 2019

Sumário

1 Introdução

Esse documento tem como objetivo especificar as características da linguagem de programação LisC. Ao longo dos capítulos subsequentes entenderemos as particularidades dessa linguagem no que diz respeito ao escopo dela, aos nomes, tipos e estrutura de dados, atribuições, expressões e mais.

A especificação dessa linguagem servirá de base para a implementação dos analisadore léxico e sintático, que serão desenvolvidas posteriormente no decorrer da disciplina. Ambos os analisadores serão implementados na linguagem de programação C. Ao fim deste documento também existirá uma seção reservada para a especificação dos tokens da linguagem.

2 Estrutura do programa

Esse capítulo tem como objetivo especificar como se da a estrutura geral de um programa em LisC. O início de um programa em LisC ocorre com a definição de uma função chamada **begin**, que retorna um *inteiro*. Sem essa função a compilação do programa da erro. Um exemplo de código de uma função **begin** pode ser visto no **código 2.1**. Em LisC temos que as funções só podem ser declaradas e definidas num escopo global, ou seja, não é permitido que uma função seja criada dentro de outra função. Note que nesse exemplo de código podemos observar que os blocos da linguagem são delimitados com a presença de chaves({}).

```
1  defun start ()(int) {
2    defint a 5;
3    echo "a = %d" (a);
4    return 0;
5  }
```

Código 2.1 – Código exemplo de uma função begin.

A presença de abre e fecha chaves define a presença de um novo escopo dentro da linguagem, variáveis que são criadas dentro desse escopo são consideradas variáveis locais, já as variáveis criadas fora de um escopo são chamadas de variáveis globais. As variáveis globais são visíveis dentro de qualquer escopo, já as locais só são vistas dentro do escopo ao qual ela foi criada, além disso a linguagem admite **mascaramento** de variáveis. Parâmetros de funções tem como escopo todo o bloco da função.

Também no código 2.1 podemos ver o primeiro exemplo da criação de uma variável. A variável criada foi uma variável do tipo inteiro. A declaração de uma variável se dá a partir da junção da palavra **def** com o tipo da variável em questão, como nesse caso tivemos uma variável inteira, a definiçao da mesma foi feita utilizando a palavra reservada **defint**. Note que a vinculação de tipos de dados às variáveis é estática, sendo assim o tipo é definido explicitamente pelo programador no momento da declaração da variável.

Toda instrução da linguagem LisC deve ser terminada com o caractere **";"**. E os comentários da linguagem são apenas comentários de linha e são feitos a partir dos caracteres **"//"**. As instruções da linguagem devem estar dentro de escopos de funções. No escopo global só é permitido a declaração ou definição de funções e variáveis.

3 Identificadores

Os identificadores em LisC são declarados como na maioria das outras linguagens, iniciando com uma letra e seguido ou não de uma ou mais letras e dígitos. Em LisC os identificadores também são sensíveis à capitalização, além de não possuir limite de caracteres. A definição de letra e dígito pode ser dada como segue:

Letra :pertence ao conjunto de caracteres $[A - Z] \cup [a - z]$

Dígito :pertence ao conjunto de caracteres $[0 - 9]$

Dessa forma temos que os identificadores em LisC podem ser representados pelo autômato cuja expressão regular é dada por $[\mathbf{A-Za-z}]([\mathbf{A-Za-z}]|[\mathbf{0-9}])*$. Note que os identificadores não podem fazer parte do conjunto das palavras reservadas da linguagem. No **capítulo** 4 vamos saber quais são as palavras reservadas da linguagem.

4 Palavras-reservadas

As palavras reservadas da linguagem podem ser vistas na tabela 1.

Tabela 1 – Tabela contendo todas as palavras reservadas da linguagem.

defun	defint	defchar	deffloat	defbool
if	else	int	for	while
echo	return	break	continue	nil
true	false	char	float	bool

5 Tipos de dados

A linguagem LisC possui os seguintes tipos de dados: inteiro, ponto flutuante, caractere e booleano. E possui as seguintes estruturas de dados: cadeia de caracteres e arranjos unidimensionais. No capítulo **??** veremos com mais detalhes como atribuir valor as variáveis que serão vistas nesse capítulo, o que envolve os tipos de operação.

5.1 Tipos de dados primitivos

Nessa seção iremos falar a respeito de cada um dos quatro tipos de dados primitivos existentes.

5.1.1 Inteiro

As variáveis do tipo inteiro representam os números inteiros. Podemos ver no **código** 5.1 como é feita a declaração de uma variável desse tipo. A palavra reservada que representa esse tipo de variável é a palavra **int**.

```
1  defint <nome>;
```

Código 5.1 – Declaração de uma variável inteira

As constantes literais de uma variável inteira estão contidas no intervalo de {-2.147.483.648, +2.147.483.647}. Temos como exemplo os seguintes valores: [-2, -40, 0, 55, 12].

5.1.2 Ponto flutuante

As variáveis do tipo ponto flutuante representam os números reais. Podemos ver no **código** 5.2 como é feita a declaração de uma variável desse tipo. A palavra reservada que representa esse tipo de variável é a palavra **float**.

```
1  deffloat <nome>;
```

Código 5.2 – Declaração de uma variável tipo ponto flutuante.

As constantes literais de uma variável float estão contidas no intervalo de {-3.4e+38, 3.4e+38}. Temos como exemplo os seguintes valores: [-2.2, -40.5, 0, 55.0, 12.1].

5.1.3 Booleano

p As variáveis do tipo booleano representam apenas dois valores: verdadeiro ou falso. Podemos ver no **código** 5.3 como é feita a declaração de uma variável desse tipo.

A palavra reservada que representa esse tipo de variável é a palavra **bool**.

```
1  defbool <nome>;
```

Código 5.3 – Declaração de uma variável booleana

As contantes literais booleanas podem ser somente: **true** ou **false**. Desse modo não existe um intervalo associado a esse tipo de variável. Em LisC qualquer valor diferente de zero é considerado verdadeiro e qualquer valor igual a zero é considerado falso.

5.1.4 Caractere

As variáveis do tipo caractere representam os caracteres alfanuméricos da tabela **ASCII**. Podemos ver no **código** 5.4 como é feita a declaração de uma variável desse tipo. A palavra reservada que representa esse tipo de variável é a palavra **char**.

```
1  defchar <nome>;
```

Código 5.4 – Declaração de uma variável caractere

As contantes literais do tipo de variável caractere estão contidas no pintervalo de [0, 127]. Temos como exemplo os seguintes valores: "a", "b", "c".

5.2 Estrutura de dados

Essa seção irá tratar dos tipos de dados que representam estruturas de dados dentro da linguagem.

5.2.1 Arranjos unidimensionais

As variáveis do tipo arranjos unidimensionais podem ser de qualquer um dos tipos primitivos já apresentados na **seção** 5.1. Esse tipo de variável é muito utilizada para guardar uma quantidade de elementos de uma variável do mesmo tipo. Note que a declaração é basicamente a mesma da declaração das variáveis primitivas, a única diferença é a existência de colchetes para definir qual será a quantidade exata dos elementos a serem guardados. Perceba, portanto, que a quantidade de elementos é estática e definida em tempo de compilação, de modo que o tamanho não pode ser alterado após ter sido declarada e/ou definida. Temos no **código** 5.5 um exemplo da declaração desse tipo de variável para cada um dos tipos primitivos.

```
1  defint numeros[5];
2  defchar letras[10];
3  defbool binarios[15];
4  deffloat taxas[4];
```

Código 5.5 – Declaração de uma variável do tipo arranjo unidimensional.

As constantes literais desse tipo de variável não possuem um intervalo de valores, visto que esse tipo de análise não faz muito sentido. Exemplos desse tipo de variável podem ser: [1,2,3], [true,false,true,false], [2.3, 1.6, -1.2].

O limite inferior de um arranjo unidimensional é 0, e o limite superior é dado pelo tamanho - 1. Desse modo, o acesso a um elemento específico dentro do arranjo pode ser feito a partir do **código** 5.6.

```
1  defint numeros[5] {1,2,3,4,5}
2  echo "acessando indice %d do arranjo numeros, valor: %d" (2, numeros[1])
```

Código 5.6 – Acessando elemento dentro de um arranjo unidimensional

5.2.2 Cadeia de caractere

As variáveis do tipo cadeia de caractere são basicamente um arranjo unidimensional de uma variável primitiva do tipo *char*. Esse tipo de estrutura também é conhecido como *string*. Temos no **código** 5.7 um exemplo da declaração de uma variável do tipo cadeia de caractere com o tamanho definido e outra com o tamanho sendo calculado a partir do valor associado a variável no momento da definição.

```
1  defchar palavra[10];
2  defchar texto[] = "Saudosa maloca";
```

Código 5.7 – Declaração de variáveis do tipo cadeia de caractere.

Note que na variável **texto** o tamanho alocado para ela foi de 15 caracteres, sendo 14 para o texto em sí(**Saudosa maloca**) e mais um para marcar o fim da string, que seria o caracter "\0". Para strings, o caracter considerado nulo é o caracter "\0"e geralmente nas linguagens é utilizado para marcar o fim de uma string.

As formas de acesso a caracteres específicos de uma cadeia de caracteres é feito da mesma que em arranjos unidimensionais.

5.3 Equivalência de tipos

5.3.1 Coerções admitidas

Os tipos de coerções admitidas na linguagem de programação LisC são relacionadas apenas as variáveis do tipo primitiva. As coerções possíveis podem ser visualizadas na **tabela** 2. A linguagem não tem nenhuma forma explícita de conversão de tipos.

Tabela 2 – Coerções admitidas em LisC

De	Para
Inteiro	String, Ponto flutuante, Caracter, Booleano
Ponto flutuante	String, Inteiro
Booleano	String
Caracter	String

5.3.2 Verificação de tipo

Para garantir que os tipos em LisC sejam respeitados, é feita uma verificação em tempo de compilação. Essa verificação é feita tanto em operações entre variáveis, para garantir que os tipos são compatíveis, como na verificação dos tipos de parâmetros de funçoes.

5.3.3 Tipagem

A linguagem LisC é fortemente tipada.

5.4 Constantes com nome

A linguagem LisC não admite constantes com nomes.

6 Atribuições e expressões

Nesse capítulo iremos tratar a respeito das atribuições e das expressões presentes em LisC. Será visto como que as atribuições são feitas e entenderemos um pouco melhor os tipos de expressões presentes na linguagem, sua precedência, quais tipos de variáveis estão associados a operação e etc. Primeiro falaremos sobre as atribuições na **seção** 6.1 e em seguida veremos a respeito das expressões na **seção** 6.2.

6.1 Atribuições

As atribuições são responsáveis por associar algum valor a uma variável. No **código** 6.1 podemos ver alguns exemplos de atribuições em LisC.

```
1  defint age 15;
2  set age ((age + 15) - 20);
3  deffloat lado, area_quadrado, area_triangulo;
4  set lado (5);
5  set area_quadrado (lado * lado);
6  set area_triangulo (area_quadrado - 1);
7  defbool x (area_quadrado > area_triangulo);
8  defbool res (!x);
```

Código 6.1 – Exemplos de atribuições.

Note que nesses exemplos as atribuições foram feitas de várias formas. No momento da declaração de uma variável é possível definir de imediato um valor inicial. Após a definição da variável, qualquer atribuição é feita a partir da palavra reservada *set*. Note que a expressão que será atribuída a variável deve estar entre parênteses.

Uma variável pode ter seu valor alterado a partir de uma constante literal, de uma expressão aritmética, relacional ou lógica, a depender do tipo da variável, retorno de função ou até mesmo de uma concatenação de cadeia de caracteres, caso a variável seja do tipo cadeia de caracteres.

6.2 Expressões

Existem quatro tipos diferentes de expressões em LisC: aritméticas, relacionais, lógicas e de concatenação. As seções a seguir vão especificar as características de cada uma das expressões disponíveis.

6.2.1 Aritmética

As expressões aritméticas estão relacionadas apenas as variáveis do tipo numéricas, ou seja, *int* e *float*. Em LisC temos somente os operadores abaixo, que estão listados de cima para baixo, do maior para o menor na ordem de precedência.

Negativo : símbolo -", é um operador unário.

Divisão e multiplicação : símbolo "/"e "*", ambos são operadores binários.

Soma e subtração : símbolo "+"e -", ambos são operadores binários.

Em expressões aritméticas temos que o operando mais a esquerda é chamado de operando alvo, ou seja, é ele quem vai ditar qual será o tipo final do resultado da operação. Sendo assim, caso o operando mais a esquerda seja do tipo *int*, mesmo que o segundo operando seja do tipo *float* o resultado da operação será um valor do tipo *int*.

6.2.2 Relacional

As expressões relacionais tem como resultado um valor sempre do tipo booleano. Podemos ver abaixo os tipos de operadores relacionais disponíveis.

Igualdade : símbolo é o "=="

Desigualdade : símbolo é o "!="

Menor que : símbolo é o «"

Maior que : símbolo é o »"

Menor ou igual que : símbolo é o «="

Maior ou igual que : símbolo é o »="

Para tipos booleanos, como é um tipo de variável que só tem dois valores, só faz sentido pensar nos operadores de igualdade e desigualdade. Contudo, com relação aos outros tipos de variáveis podemos criar uma expressão relacional com qualquer um dos operadores disponíveis.

Nas operações aritméticas tinhamos que mesmo que os operandos envolvidos não fossem do mesmo tipo, existia uma coerção automática para concluir o resultado da operação. Contudo, para expressões relacionais esse tipo de coerção não existe e resulta em erros. O motivo é trivial de perceber, não faz sentido relacionar duas variáveis de tipos diferentes.

6.2.3 Lógica

As expressões lógicas são utilizadas apenas quando os seus operandos são booleanos. Podemos ver abaixo os tipos de operandos lógicos disponíveis.

Negação : símbolo "!", unário

Conjunção : símbolo "&&", binário

Disjunção : símbolo |", binário

Os operadores foram dispostos em ordem de precedência, do maior para o menor.

6.2.4 Concatenaçao

Voltando daqui...