COMPILADORES

João Vitor Alves Rocha e Leonardo Oenning Spilere

MANUAL DE LINGUAGEM E AÇÕES SEMÂNTICAS

1. **Tipos**

Para declararmos variáveis e constantes, precisamos primeiramente entender os **<tipos>** disponíveis, sendo eles inteiros, reais, strings caracteres e arrays.

**Inteiros**

Inteiros são números não decimais positivos e utilizam a palavra reservada ‘integer’.

**Exemplo:** 420

**Reais**

Reais são números decimais positivos e utilizam a palavra reservada ‘real’, onde o valor decimal é separado por um ‘.’ (ponto final) e tem no máximo duas casas decimais.

**Exemplo:** 6.9 1.23

**Caracteres**

Caracteres são letras, números ou símbolos envolvidos por aspas simples (‘’) que serão tratados como texto. Cada char pode conter apenas um caracter por si só.

**Exemplo:** ‘F’ ‘9’ ‘?’

**Strings**

Strings são letras, números ou símbolos envolvidos por aspas duplas (“”’) que serão tratados como texto. Cada string pode conter no máximo 32 caracteres por si só.

**Exemplo:** “1ͦ colocado: Fernando Alonso”

**Arrays**

Arrays são listas de tipos. Um array pode conter uma ou mais listas, porém de apenas um tipo, e os tipos permitidos são apenas os previamente citados (não é possível criar um array de arrays).

Para criar um array utilizamos a palavra reservada ‘array’ seguido da posição inicial e posição final representados por **<posição>** (sendo esses números inteiros e positivos) dentro de [] (colchetes) e separados pela palavra reservada ‘..’, seguido pela palavra reservada ‘of’ e **<tipo>** (tipo do array).

**Sintaxe:**

array [<posição> .. <posição>] of <tipo>

**Exemplos:**

array [0 .. 4] of integer

array [1 .. 10] of string

1. **Comentários**

Os comentários são inseridos no código para informar algo para o programador, ou deixar alguma anotação sobre o código, este trecho é ignorado pelo compilador.

Para comentar um código em uma linha, devem ter ‘//’ no início, e para comentar em bloco deve-se iniciar com ‘/\*’ e no final do código adicionar ‘\*/’;

**Sintaxe:**

//<comentário>

/\*<comentário>

<comentário>

<comentário>\*/

**Exemplos:**

//linha comentada

/\*este

bloco está

comentado\*/

1. **Iniciando o programa**

Todo programa deve iniciar com a palavra reservada ‘program’ seguida do nome do programa definido pelo desenvolvedor e finalizado com ‘;’ (ponto e vírgula).

Então é inserido o <bloco de código> e finalizamos o programa com ‘.’ (ponto final)

**Sintaxe:**

program <identificador>;

<bloco>

.

**Exemplo:**

program NomeDoMeuPrograma;

<bloco de código>

.

//note o ponto final após o bloco de código

1. **Bloco de código**

O bloco de código pode conter as declarações de ‘procedure', constante ou variável e obrigatoriamente um corpo.

1. **Corpo**

Deve iniciar com a palavra reservada ‘begin’ seguido opcionalmente por N ‘comando’ (onde N é número de comandos desejados), separados por ‘;’ (ponto e vírgula), e terminados em ‘;’ ‘end’.

**Sintaxe:**

begin

<procedure>;

<procedure>;

<constante>;

<variável>;

<corpo>;

end

begin

<procedure>;

<corpo>;

end

begin

<corpo>;

end

**Exemplo:**

begin

//declaração de procedure

//declaração de procedure

//declaração de variáveis

//corpo

end

1. **Declaração de Procedure**

A procedure é um bloco de código que realiza uma tarefa específica, e pode ser chamado em diferentes partes do código, facilitando a manutenção e a reutilização.

Inicialmente podemos simplesmente utilizar o ‘procedure’, ‘identificador’ e **<corpo>** ‘;’.

**Sintaxe:**

procedure <identificador> begin <comando>; end;

**Exemplo:**

procedure meuProcedure begin write(\_literal\_); end;

Após o identificador, é possível adicionar parâmetros, com o seus respectivos tipos, englobados por “( )”;

**Sintaxe:**

procedure <identificador> (<identificador>, <identificador> : <tipo>; <identificador>,<identificador>: char) begin <comando>; end;

**Exemplo:**

procedure meuProcedure (identificador1, identificador2: string; identificador3, identificador4: char) begin write(\_literal\_); end;

Ao declarar procedures, também é possível adicionar declarações de variáveis como mostrado anteriormente.

**Sintaxe:**

procedure <identificador> () declaravariaveis <identificador> : <tipo> ; <identificador> : <tipo>; begin <comando>; end;

**Exemplo:**

procedure nomeProcedure declaravariaveis variavel1 : char ; variavel2 : string; begin write(\_literal\_); end;

No final da declaração de procedure, após o ‘;’ é possível declarar outra ‘procedure’, com a possibilidade de utilização dos mesmos atributos explicados anteriormente.

**Sintaxe:**

procedure <identificador> begin <comando>; end; procedure <identificador> begin <comando>; end;

**Exemplo:**

procedure nomeProcedure1 begin write(\_primeira procedure\_); end; procedure nomeProcedure2 begin write(\_segunda procedure\_); end;

1. **Declaração de Constante**

Uma constante é um valor que, uma vez definido não pode ser alterado durante a execução do programa, são utilizadas para representar valores que devem permanecer iguais no código.

Para declarar uma constante da forma mais simples podemos iniciar com ‘const’,‘identificador’, ‘=’, <tipo> e ‘;’.

**Sintaxe:**

const <identificador> = <tipo> ;

**Exemplo:**

const minhaConstante = integer; minhaConstante = string;

Sendo possível também adicionar mais uma constante aproveitando o ‘const’ inicial, após o ‘;’. Inicialmente com ‘identificador’, ‘=’, <tipo>, ‘;’, e novamente, podendo adicionar mais constantes desta mesma forma.

**Sintaxe:**

const <identificador> = <tipo> ; <identificador> = <tipo>;<identificador> = <tipo> ; <identificador> = <tipo>; <identificador> = <tipo>;

**Exemplo:**

const minhaConstante = interger; segundaConstante = string;minhaConst = interger; segundaConst = string; terceiraConst = char;

1. **Declaração de Variáveis**

As variáveis podem receber dados, mas diferentemente das constantes, o valor destas podem mudar durante a execução do programa, sendo utilizadas para manipulação das informações.

Para declará-las iniciamos com ‘declaravariaveis’, ‘identificador’, sendo possível adicionar mais identificadores concatenados com ‘,’, seguindo com ‘:’, <tipo>, ‘;’, e podendo ser declarado outras variáveis juntas, com ‘identificador’, ‘:’, <tipo>, ‘;’, e repetir essa mesma sintaxe para definir mais variáveis.

**Sintaxe:**

declaravariaveis <identificador> : <tipo> ; <identificador> : <tipo>;<identificador>, <identificador> : <tipo> ; <identificador> : <tipo>; <identificador> : <tipo>;<identificador>, <identificador> : <tipo> ; <identificador>, <identificador> : <tipo>; <identificador> : <tipo>;

**Exemplo:**

declaravariaveis primeiraVar : string; segundaVar: integer; terceiraVar: char;primeiraVar, segundaVar: string; terceiraVar: real;primeiraVar, segundaVar: string; terceiraVar, quartaVar: real;

1. **Comandos**

No trecho de comandos poderemos ter opcionalmente estruturas de **repetição**, estruturas **condicionais**, métodos de **chamado**, **entrada** e **saída** de dados, mas para utilizá-los precisamos entender as **expressões** que estarão dentro deles.

1. **Expressões**

As expressões são uma combinação de valores, variáveis, operadores e chamadas de funções, que serão utilizadas em vários lugares.

Deve ser iniciada com o **termo**, constituído pelo **fator**, que pode ser um número inteiro, identificador, nome string, nome char, número real ou até mesmo uma expressão, englobada com ‘( )’ (parênteses), e tendo a opção de ser seguido pela **repetição do termo** que inicia pelos operadores ‘\*’, ‘/’, ‘and’ e com isso seguido novamente de um **fator** podendo também haver N **repetições do termo,** e também a **repetição da expressão.**

Após o termo ser montado, podemos adicionar de forma opcional a **repetição da expressão**, sendo constituída de ‘+’ ou ‘-’ ou ‘or’, seguido do **termo,** explicado anteriormente, e novamente tendo a possibilidade de adicionar mais uma **repetição da expressão.**

**Sintaxe: Termo**

<fator> <operador> <fator>

<fator> <operador> <fator> <operador> <fator>

**Exemplo: Termo**

//podemos ver abaixo exemplos apenas com o fator, sem a repetição do termo e a repetição da expressão

“nomeJoao”

5

‘j’

//podemos ver abaixo exemplos com o fator e a repetição do termo e a repetição da expressão

1 \* “nomeJoao”

1 and “nomeJoao”

1 / “nomeJoao”

1 + “nomeJoao”

1 - 1.2 + “nomeLeo” or ‘j’

1 or 2.54

//podemos ver abaixo exemplos com o fator e a repetição do termo duas vezes

1 / “nomeJoao” \* ‘j’

1 / “nomeJoao” \* 1.4

Pode ser adicionado também a **repetição da expressão simples** iniciada com os operadores ‘=’, ‘<’, ‘>’, ‘<=’, ‘>=’ ou ‘<>’ seguido necessariamente da **expressão simples** que pode ser iniciada com ‘+’ ou ‘-’, seguido do **termo**, anteriormente explicado, e da **repetição da expressão,** também explicado anteriormente.

**Exemplo: Repetição da expressão simples**

= “nomeJoao”

< “nomeJoao”

<= + “nomeJoao” - 1.2 + “nomeLeo” or ‘j’

Assim com as definições anteriores podemos montar expressões completas, conforme necessário, seguindo a ordem **termo,** obrigatório, **repetição da expressão,** opcional, e a **repetição da expressão simples,** opcional.

**Sintaxe:**

<fator> <operador \* / and> <fator> <operador + - or> <termo> <operador matemático> <operador de comparação>

**Exemplo:**

1 / “nomeJoao” \* ‘j’ + “nomeJoao” < “nomeJoao”

1. **Entrada dados**

Para utilizar o recurso de entrada deve-se utilizar o comando ‘read’, seguido por ‘( )’ informando a <váriavel> as ser lida dentro do parênteses, e pode ser intercalado com ', ' caso tenha mais variáveis.

**Sintaxe:**

read(<variável>);

read(<variável>,<variável>,<variável>);

**Exemplo:**

// O valor digitado será armazenado na variável ‘x’;

read(x);

read(x, y, z);

1. **Saída de dados**

O comando ‘write’ é utilizado para exibir um valor <literal> ou uma <expressão>, e pode também ser intercalado por vírgula para exibir mais de um item.

**Sintaxe:**

write(<literal>);

write(<expressão>);

write(<literal>, <literal>, <expressão>);

**Exemplo 1:**

write(\_literal\_);

write(\_literal\_,\_outro literal\_, z);

write(\_o valor de x é \_, x, \_o valor de y é\_, y);

1. **Métodos de chamado**

Para chamar uma procedure é necessário utilizar a palavra reservada ‘chamaprocedure’ seguido por <identificador> e opcionalmente os parâmetros da procedure.

Os parâmetros são variáveis e devem estar presentes devem estar dentro de ‘( )’ e separados por ‘, ’ caso haja mais de uma parâmetro.

**Sintaxe:**

chamaprocedure <identificador> ()

chamaprocedure <identificador> (<parâmetro>)

chamaprocedure <identificador> (<parâmetro>, <parâmetro>)

**Exemplo:**

chamaprocedure NomeDaProcedure()

chamaprocedure NomeDaProcedure(Variavel)

chamaprocedure OutroProcedure(Variavel, OutraVariavel)

1. **Estrutura condicional: IF**

Avalia a condição especificada, e executa a primeira linha de código caso seja verdadeira, caso seja falsa, encaminha-se para a próxima condição e executa segunda linha de código, caso seja verdadeira, e assim por diante.

Para utilizar, inicia-se com a palavra ‘if’, <expressão> englobada com ‘[ ]’, ‘then’, ‘begin’, <comando>, ‘end’, e opcionalmente o ‘else’, ‘begin’, <comando> e ‘end’.

**Sintaxe:**

if [<expressao>] then begin <comando> end

if [<expressao>] then begin <comando> end else begin <comando> end

**Exemplo:**

if[inteiro > outro] then begin write(\_mensagem literal\_) end

if[inteiro < outro] then begin write(\_mensagem literal\_) end else begin write(\_outro literal\_) end

1. **Estrutura de repetição: FOR**

O comando ‘for’ é utilizado para criar um loop que itera sobre uma faixa de valores. O bloco de código dentro do for é executado para cada valor iterado. Deve ser iniciado com a palavra ‘for’ seguido com o **<identificador>** ‘=’ **<expressão>**, englobado por ‘[ ]’ (colchetes), continuado com ‘to’ **<expressão>**, englobado com ‘[ ]’, e logo após, ‘do’ ‘begin’ **<comando>** ‘end’.

**Sintaxe:**

for [ <identificador> = <expressão> ] to [ <expressão> ] do begin <comando> end

**Exemplo:**

for [ numeroInicial = 1 ] to [ \_pessoaJoao\_ ] do begin write(\_literal\_) end

1. **Estrutura de repetição: WHILE**

Recebe uma condição para executar um bloco de código, e é interrompida quando a condição é falsa, que é verificada em antes de cada iteração.

**Sintaxe:**

while [<expressão>] do begin <comando> end

**Exemplo:**

while [numero < 10] do begin write(\_ainda dentro do loop\_) end

**Estrutura de repetição: REPEAT**

Executa o bloco de código pelo menos umas vez, e continua a execução até que a condição especificada seja verdadeira, que é verificada em cada iteração.

**Sintaxe:**

repeat <comando> until [<expressão>]

**Exemplo:**

repeat write(\_dentro do loop\_) until [numero < 10]

1. **Palavras Reservadas**

* write
* while
* until
* to
* then
* string
* repeat
* real
* read
* program
* procedure
* or
* of
* integer
* if
* identificador
* for
* end
* else
* do
* declaravariaveis
* const
* char
* chamaprocedure
* begin
* array
* and

1. **Regras léxicas**

* Nome de identificador devem ter no máximo 20 caracteres;
* Nome de identificador só podem contar letras ou números, e não devem ser idênticas a uma das palavras reservadas;
* Nome de identificador não devem conter caracteres especiais nem espaço em branco;
* Nome de identificador deve sempre começar com uma letra;
* Dados do tipo ‘integer’ devem ter no máximo 5 números;
* Os dados integer e reais não devem ser negativos
* Deve-se utilizar ‘.’ para separar os decimais do tipo ’real’;
* Dados do tipo ‘real’ devem ter no máximo 5 números antes do ponto, e 2 após o ponto;
* Não deve ser nomeado programa, constantes, variáveis ou procedures como qualquer uma das palavras reservadas;
* O tipo ‘literal’ deve ser utilizado entre ‘\_’ (*underscore*);
* Dados do tipo ‘char’ devem conter apenas 1 caractere;
* Dados do tipo ‘char’ devem ser definidos entre aspas simples;
* Dados do tipo ‘string’ devem conter no máximo 20 caractere;
* Dados do tipo ‘string’ devem ser definidos entre aspas duplas;
* Comentários em linha devem iniciar com ‘//‘, desabilitando toda a linha a partir do símbolo;
* Comentários em bloco devem iniciar com ‘/\*’ e finalizar com ‘\*/’, desabilitando todo o conteúdo entre os símbolos.

1. **Erros léxicos**

* Identificadores declarados contendo mais de 20 caracteres;
* Identificadores declarados contendo caracteres especiais;
* Identificadores declarados não iniciando com uma letra;
* Número do tipo integer com mais de 5 dígitos;
* Número do tipo real usando vírgula ao invés de ponto como separador decimal;
* Número do tipo real com mais de 5 dígitos antes do ponto ou mais de 2 dígitos após o ponto;
* Dados do tipo real ou integer serem negativos;
* Uso de mais de um caractere em dados do tipo char;
* String com mais de 20 caracteres;

**Ações semânticas**;

**1 - Verificação de variáveis não declaradas:**

Verifique se uma variável é usada antes de ser declarada. Isso evita erros ao tentar acessar uma variável inexistente.

def check\_variable\_declared(variables, var\_name):

if var\_name not in variables:

return f"Erro: variável '{var\_name}' não foi declarada."

return "OK"

**2 - Tipos de dados em operações:**

Certifique-se de que uma operação entre dois valores faz sentido com seus tipos. Por exemplo, soma entre números ou concatenação entre strings.

def check\_data\_types(var1, var2, operation):

if operation == "+":

if isinstance(var1, (int, float)) and isinstance(var2, (int, float)):

return "OK"

elif isinstance(var1, str) and isinstance(var2, str):

return "OK"

else:

return "Erro: tipos incompatíveis para soma."

**3 - Divisão por zero:**

Garanta que o divisor em uma operação de divisão não seja zero.

def check\_division\_by\_zero(divisor):

if divisor == 0:

return "Erro: divisão por zero."

return "OK"

**4 - Declaração de variáveis duplicadas:**

Verifique se uma variável está sendo declarada mais de uma vez.

def check\_duplicate\_declaration(variables, var\_name):

if var\_name in variables:

return f"Erro: variável '{var\_name}' já foi declarada."

variables.add(var\_name)

return "OK"

**5 - Verificação de escopo:**

Verifique se uma variável está sendo usada fora de seu escopo, como uma variável local que não é acessível fora de uma função.

def check\_scope(variable\_scope, var\_name, current\_scope):

if var\_name not in variable\_scope.get(current\_scope, set()):

return f"Erro: variável '{var\_name}' fora do escopo."

return "OK"