

UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL
INSTITUTO DE COMPUTAÇÃO - IC

ERIC DOS SANTOS COELHO

ESPECIFICAÇÃO DA LINGUAGEM ESC

Trabalho apresentado à disciplina de
Compiladores, ministrada pelo professor Alcino
Dall Igna Junior.

MACEIÓ
2019.1

Sumário

Estrutura geral de um programa	3
Nomes	3
Tipos e estruturas de dados	3
Forma de declaração	3
Operadores	4
Constantes literais	4
Declarações	4
Cadeias de caracteres	4
Arranjos	4
Equivalência de tipos e coerção	4
Atribuição e expressões	5
Precedência	5
Associatividade	5
Sintaxe e exemplo de estruturas de controle	5
Comando de seleção	5
Comandos de iteração	6
Controle lógico	6
Controle por contador	6
Desvio incondicional	6
Instruções de entrada e saída	6
Subprogramas	7
Métodos de passagem de parâmetros	7
Exemplos	8
Alô mundo	8
Série de Fibonacci	8
Shell sort	9
Especificação dos tokens	10
Enumeração com as categorias dos tokens	10
Categorias simbólicas e expressões regulares	10
Expressões regulares auxiliares	12

Estrutura geral de um programa

A linguagem admite escopo global e as variáveis são declaradas ao especificar o tipo e o nome em qualquer parte do programa. Funções são precedidas da palavra 'func' e são declaradas especificando o tipo de retorno, o nome e a lista de parâmetros.

A execução é iniciada a partir da função *init()* e as funções são definidas em qualquer parte especificando o tipo de retorno, o nome e a lista de parâmetros.

Nomes

Os nomes são sensíveis à caixa e tem tamanho máximo de 100 caracteres alfanuméricos iniciando com uma letra.

Os nome são reconhecidos pela expressão regular `[_a-zA-Z][_a-zA-Z0-9]*`, são permitidos caracteres alfanuméricos e underscore exceto o primeiro que não pode ser um dígito.

Tipos e estruturas de dados

Forma de declaração

Os tipos de dados predefinidos e suas operações suportadas são:

Tipo	Declaração	Operações
Inteiro	<code>int nome;</code>	Aritméticas e relacionais
Ponto flutuante	<code>float nome;</code>	Aritméticas e relacionais
Caractere	<code>char nome;</code>	Relacionais
Booleano	<code>bool nome;</code>	Relacionais de igualdade e diferença e lógicas
Cadeias de caracteres	<code>string nome;</code>	Relacionais e concatenação
Arranjos	<code>tipo nome[tamanho];</code>	Concatenação

Operadores

Aritméticos	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> (módulo), <code>-</code> (unário)
Relacionais	<code>==</code> , <code>!=</code> , <code><=</code> , <code>>=</code> , <code><</code> , <code>></code>
Lógicos	<code>!</code> , <code>and</code> , <code>or</code>
Concatenação	<code>++</code>

Constantes literais

```
int          (\d+)
float        (\d+)\.(\d+)
char         \'
bool         (true)|(false)
string       "(.*)"
Arranjos     { el1, el2,... }
```

Declarações

Os tipos inteiro, ponto flutuante, caractere, booleano e string são declarados seguindo o formato `tipo nome;`

As constantes nomeadas são precedidas pela palavra reservada `const` e em seguida deve ser declarada a variável.

Cadeias de caracteres

Para as cadeias de caracteres deve ser utilizadas aspas (`"`) para atribuição:

```
string nome = "valor";
```

Arranjos

Os arranjos são declarados na forma `tipo nome[tamanho];`

Os elementos dos arranjos são referenciados por índices listados entre colchetes e são armazenados em posições contíguas na memória.

Equivalência de tipos e coerção

A equivalência de tipos é por nome e é permitido conversões de tipo explícitas (`cast`). A coerção admitida é de `int` para `float`.

Atribuição e expressões

A atribuição é dada pelo comando `var_alvo = expressao` onde a expressão do lado direito é atribuída à variável alvo do lado esquerdo. Os tipos das operações são definidos de acordo com a variável alvo.

Os operadores `and` e `or` são avaliados em curto-circuito.

Precedência

Precedência da mais alta para a mais baixa

```
!, - unário
*, /, %
+ e - binários
++
<, >, <=, >=, ==, !=
and, or
```

Associatividade

```
Esquerda:  *, /, %, + e - binários
            <, >, <=, >=, ==, !=
            and, or
Direita:    !, - unário
```

Sintaxe e exemplo de estruturas de controle

Todos os blocos devem possuir chaves (`{}`)

Comando de seleção

O comando de seleção é o `if` que seleciona um caminho baseado em cada condição. Para mais de uma condição é usado o `else if` e o bloco opcional `else` que é selecionado caso nenhuma das condições sejam atendidas.

```
if(condição) {
    ...
} else if(condição) {
    ...
} else {
    ...
}
```

Comandos de iteração

Controle lógico

Enquanto a condição não for verdadeira o bloco de código é executado.

```
while (condição){  
    ...  
}
```

Controle por contador

A variável do laço é especificada em *var_int* e é incrementada ou decrementada pela constante inteira *val_increm*. O intervalo discreto de valores é definido em *in (valor_inicial, valor_final)*.

```
for var_int in (valor_inicial, valor_final) step val_increm{  
    ...  
}
```

Desvio incondicional

break

Permite encerrar um loop

```
while (condição){  
    ...  
    break;  
    ...  
}
```

Instruções de entrada e saída

A instrução de leitura da entrada padrão é realizada especificando as variáveis em que serão atribuídos os valores lidos.

```
input var1, var2,...;
```

Na instrução de saída padrão deve ser especificado o texto formatado e as variáveis ou constantes que vão ser substituídas

```
print "%2d %.2f" var1, cons1;
```

Subprogramas

As funções são declaradas no corpo do programa e seguem a forma:

```
tipo nome(parâmetros) {  
    ...  
}
```

Os procedimentos são declarados como funções. A palavra reservada *proc* é usada para declarar os procedimentos.

```
proc nome(parâmetros) {  
    ...  
}
```

Métodos de passagem de parâmetros

As funções implementam o modo de entrada e saída, os parâmetros são passados por valor-resultado. Funções não podem ser passadas como parâmetro.

Os procedimentos são implementados no modo de entrada.

Exemplos

Alô mundo

```
int init(){
    print "Alô mundo";
    return 0;
}
```

Série de Fibonacci

```
proc fibonacci(int valor_limite){
    int a, b, aux;

    if(valor_limite >= 0){

        print "0";

        a = 0;
        b = 1;
        aux = a + b;

        while(aux <= limite){
            print ", %d", aux;
            aux = a + b;
            a = b;
            b = aux;
        }
    }
}

int init(){
    int limite;

    print "Digite um valor limite: ";
    input limite;
    fibonacci(limite);
    return 0;
}
```


Shell sort

```
proc shellSort(int vet[], int size){

    int i , j , value;
    int gap = 1;

    while(gap < size) {
        gap = 3*gap+1;
    }

    while (gap > 0) {
        for i in (gap, size) step 1 {
            value = vet[i];
            j = i;
            while (j > gap-1 and value <= vet[j - gap]) {
                vet[j] = vet [j - gap];
                j = j - gap;
            }
            vet[j] = value;
        }
        gap = gap/3;
    }
}

int init(){

    int i, tam;

    input tam;
    int vet[tam];

    for(i = 0; i < tam; 1){
        input vet[i];
        print "%d ", vet[i];
    }
    shellSort(vet, tam);

    for(i = 0; i < tam; 1){
        print "%d ", vet[i];
    }
}
```

Especificação dos tokens

A linguagem em que os analisadores léxico e sintático serão implementados será C++

Enumeração com as categorias dos tokens

```
enum class Category {  
    Init=1, Integer, Float, Char, String, Boolean,  
    PtVg, Pt2, Vg, AbPar, FePar, AbCol, FeCol, AbChav, FeChav,  
    Procedure, Func, Return, Break,  
    Input, Print,  
    For, In, Step, While, If, ElseIf, Else,  
    OpEq, OpMaior, OpMenor, OpMaiorEq, OpMenorEq, OpDifer, OpAnd, OpOr, OpNot,  
    OpMais, OpMenos, OpMult, OpDiv, OpMod,  
    OpAtr, OpConcat,  
    Id, CteInt, CteFloat, CteChar, CteBool, CteStr, Eof  
};
```

Categorias simbólicas e expressões regulares

Init	<i>init</i>
Integer	<i>int</i>
Float	<i>float</i>
Char	<i>char</i>
String	<i>string</i>
Boolean	<i>bool</i>
PtVg	<i>;</i>
Pt2	<i>:</i>
Vg	<i>,</i>
AbPar	<i>(</i>

FePar)
AbCol	[
FeCol]
AbChav	{
FeChav	}
Procedure	<i>proc</i>
Func	<i>func</i>
Return	<i>return</i>
Break	<i>break</i>
Input	<i>input</i>
Print	<i>print</i>
For	<i>for</i>
In	<i>in</i>
Step	<i>step</i>
While	<i>while</i>
If	<i>if</i>
Elseif	<i>else if</i>
Else	<i>else</i>
OpEq	<i>==</i>
OpMaior	<i>></i>
OpMenor	<i><</i>
OpMaiorEq	<i>>=</i>
OpMenorEq	<i><=</i>
OpDifer	<i>!=</i>

OpAnd	<i>and</i>
OpOr	<i>or</i>
OpNot	<i>!</i>
OpMais	<i>+</i>
OpMenos	<i>-</i>
OpMult	<i>*</i>
OpDiv	<i>/</i>
OpMod	<i>%</i>
OpAtr	<i>=</i>
OpConcat	<i>++</i>
Id	<i>[_a-zA-Z][_a-zA-Z0-9]*</i>
CteInt	<i>-?(\d+)</i>
CteFloat	<i>-?(\d+)\.(\d+)</i>
CteChar	<i>\'.\'</i>
CteBool	<i>(true) (false)</i>
CteStr	<i>\"(.*)\"</i>
Eof	<i>\0</i>

Expressões regulares auxiliares

```

letras = '[a-zA-Z]'
dígitos = '[0-9]'
símbolos = '[.,;!\+-*\|/%<>=() [] {} " ' ]'
```