UNIOESTE – Universidade Estadual do Oeste do Paraná
Campus de Cascavel PR - Rua Universitária, 1619
(45) 3220-3000

Bacharelado em Ciência da Computação - CCET
Colegiado de Ciência da Computação
(45) 3220-3191

Disciplina: **Linguagens de Programação**
Alunos: **Juliano Felipe da Silva**
Maycon de Queiroz Oliveira

21/02/2017

Descrição da linguagem de programação NOME

1 OPERADORES

1.1 OPERADORES LÓGICOS SUPORTADOS

Símbolo Utilizado	Operação Realizada	Exemplo de chamada
&	E bit a bit	C = A & B
|	OU bit a bit	C = A | B
!	Negação	C = !A
^	OU exclusivo bit a bit	C = A ^ B
<<	Shift lógico a esquerda onde o elemento a esquerda sofre a quantidade apresentada à direita do operador	C = A << B
>>	Shift lógico a direita onde o elemento a esquerda sofre a quantidade apresentada à direita do operador	C = A >> B

1.2 OPERADORES ARITMÉTICOS SUPORTADOS

Símbolo Utilizado	Operação Realizada	Exemplo de chamada
+	Soma	C = A + B
-	Subtração	C = A - B
*	Multiplicação	C = A * B
/	Divisão	C = A / B
%	Resto da divisão do elemento da direita pelo elemento da esquerda	C = A % B

1.3 OPERADORES RELACIONAIS SUPORTADOS

Símbolo Utilizado	Operação Realizada	Exemplo de chamada
==	Verdadeiro se elementos a direita e a esquerda são iguais	A == B
!=	Verdadeiro se elementos a direita e a esquerda são diferentes	A != B
<=	Verdadeiro se elemento a esquerda é menor ou igual ao elemento a direita	A <= B
>=	Verdadeiro se elemento a esquerda é maior ou igual ao elemento a direita	A >= B
<	Verdadeiro se elemento a esquerda é menor que o elemento a direita	A < B
>	Verdadeiro se elemento a esquerda é maior que o elemento a direita	A > B

1.4 EBNF

<oper_logi> → (<id> | <letr> | <digi>{<digi>}+) (& | '|' | ^ | '<<' | '>>' | !) (<id> | <letr> | <digi>{<digi>}+)

<oper_arit> → (<id> | <letr> | <digi>{<digi>}+) (+| - | * | / | %) (<id> | <letr> | <digi>{<digi>}+)

<lexp> → <id> (== | != | '<'= | '>'= | '<' | '>') (<id> | <letr> | <digi>{<digi>}+)

<oper_term>→ <id> (& | '|' | ^ | '<<' | '>>' | + | - | * | / | %) '('<term>')' | ! '('<term>')' | '('<term>')' (& | '|' | ^ | '<<' | '>>' | + | - | * | / | %) <id>

<term> → <oper_term> | <oper_logi> | <oper_arit>

<atri> → <id> = (<oper_logi> | <oper_arit> | <oper_term>);[<ql>]

2 TIPOS DE DADOS

A declaração de variáveis é explicita, com o tipo informado logo à frente da variável a ser declarada. Os possíveis tipos e suas declarações são listados abaixo.

2.1 CHR

Tamanho	1 byte
Operações Permitidas	+, -, *, /, %, &, |, !, ^, <<, >>
Chamada em impressão	%c – Imprime caractere de acordo com tabela ASCII %u – Imprime valor numérico do byte
Descrição	Valor que pode ser tanto caractere como inteiro

2.2 INT

Tamanho	4 bytes
Operações Permitidas	+, -, *, /, %, &, |, !, ^, <<, >>
Chamada em impressão	%d – Imprime valor numérico
Descrição	Valor inteiro

2.3 FLT

Tamanho	4 bytes
Operações Permitidas	+, -, *, /, %, &, |, !, ^
Chamada em impressão	%f – Imprime valor numérico
Descrição	Valor ponto flutuante

2.4 BLN

Tamanho	1 byte (usado apenas bit menos significativo)
Operações Permitidas	&, |, !, ^, <<, >>
Chamada em impressão	%b – Imprime valor numérico (1 ou 0)
Descrição	Valor para uso booleano, 1 para verdadeiro e 0 para falso

2.5 EBNF

Declaração de variáveis. Podem ser declarados vetores de até 255 dimensões.

<decl_stmt> → (chr | int | flt | bln) <id>{‘[‘<digi>{<digi>}*’]’}$^{0\sim255}$;[<ql>]

3 ESTRUTURAS DE DESVIOS

3.1 DESVIOS INCONDICIONAIS

3.1.1 GTO

Salta para uma linha do código denominada por uma label. Chamada sem parâmetros é inválida. Exemplo: gto label1

3.1.1.1 Labels

Labels são declaradas no código sendo precedidas por dois pontos (:). Exemplo: :label1

3.1.2 BRK

Para a execução do laço de repetição. Chamada com parâmetros ou sem laço é inválida. Exemplo: brk

3.2 DESVIOS CONDICIONAIS

3.2.1 IFF

Executa o código dentro do bloco posterior se a expressão neste for verdadeira. Expressões vazias são inválidas. Um bloco posterior deve ser definido usando chaves. Exemplo:

```
iff

(A < B){

}
```

3.2.2 ELS

Executa o código dentro do bloco posterior se a expressão ‘iff’ acima for falsa. Expressões são inválidas. Um bloco posterior deve ser definido usando chaves. Exemplo:

```
iff(A < B){

}

els {

}
```

3.2.3 EIF

Executa o código dentro do bloco posterior se a expressão 'iff' acima for falsa, mas a expressão neste for verdadeira. Expressões vazias são inválidas. Um bloco posterior deve ser definido usando chaves. Exemplo:

```
iff (A < B){

}

eif (A == B){

}
```

3.3 EBNF

```
<cond_stmt> → iff'('<lexp>')''{'<ql><tb><stmt>{<tb><stmt>}*'}'<ql>[(
              eif'('<lexp>')''{'<ql><tb><stmt>{<tb><stmt>}*'}'<ql> els
              '{'<ql><tb><stmt>{<tb><stmt>}*'}'<ql> | els
              '{'<ql><tb><stmt>{<tb><stmt>}*'}'<ql>)] | gto <id>;[<ql>]
<lbel>      → :<id>;[<ql>]
```

4 ESTRUTURAS DE REPETIÇÃO

4.1 FOR

Executa o código dentro do bloco posterior de acordo com as condições iniciais, final e de alteração. Um bloco posterior deve ser definido usando chaves. Exemplo:

```
for(A = 0; A < 10; A = A + 1){

}
```

4.2 WHL

Executa o código no bloco posterior enquanto a expressão for válida. Expressões vazias são inválidas. Um bloco posterior deve ser definido usando chaves. Exemplo:

```
whl(A == 0){

}
```

4.3 EBNF

```
<rept_stmt> → for'('<atri>; <lexp>; <atri>')''{'<ql><tb><stmt>{<tb><stmt>}*'}'<ql> |
              whl'('<lexp>')''{'<ql><tb><stmt>{<tb><stmt>}*'}'<ql>
```

5 DEFINIÇÕES

5.1 PALAVRAS RESERVADAS

Todos os comandos são palavras reservadas. As palavras A00 a A99 (de 00 a 99 sucedendo a letra A maiúscula) são palavras reservadas.

5.2 NOMES DE VARIÁVEIS

Os nomes de variáveis podem ter até 255 caracteres e devem iniciar com uma letra. Os nomes de variáveis são sensíveis a alteração de letras maiúsculas e minúsculas.

5.2.1 EBNF

<digi>→ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letr>→ A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
<id> → <letr>{<letr> | <digi>}$^{0\sim254}$

6 EBNF

Todo o desenvolvimento de EBNF de um programa se inicia com <begi> em 6.3.

6.1 VALORES NÃO PREVIAMENTE IDENTIFICADOS

<ql> → Quebra de linha
<tb> → Tabulação
<simb> → ! | “ | # | $ | % | & | ‘” | ‘(‘ | ‘)’ | * | + | , | - | . | / | : | ; | ‘<’ | = | ‘>’ | ? | @ | ‘[‘ | ‘]’ | _ | ^ | ´ | ` | ~ | ‘{‘ | ‘}’ | ‘|’

6.2 SUBROTINAS

<subr> → (chr | int | flt | bln) sub <id>’(‘[(chr | int | flt | bln) <id>{, (chr | int | flt | bln) <id>}*]’)’’{‘<ql><tb><stmt>‘}’<ql>
<decl_subr> → (chr | int | flt | bln) sub <id>’(‘[(chr | int | flt | bln) <id>{, (chr | int | flt | bln) <id>}*]’)’;<ql>
<list_desu> → <decl_subr>{<decl_sub>}*
<cham_subr>→ <id> = <id>’(‘[<id>{, <id>}*]’)’;[<ql>]

6.3 EBNF GERAL DA LINGUAGEM

<begi> → <exfc><ql>[<decl_stmt>{<decl_stmt>}*]<list_desu><ql>ini’{‘<ql><tb><stmt>{<tb><stmt>}*<tb><retr><ql>’}’[<ql><ql><subr>{<subr>}*]
<exfc> → $inc ‘<’<id>’>’;<ql>
<stmt> → <atri>[<stmt>] | <decl_stmt>[<stmt>] | <lbel>[<stmt>] | <rept_stmt>[<stmt>] | <cham_subr>[<stmt>] | <cond_stmt>[<stmt>] | <ql>[<stmt>] | <retr>[<stmt>] | <cmnt>[<stmt>]
<retr> → ret (<id> | <letr> | <digi>{<digi>}+)
<cmnt> → #{(<simb> | <letr> | <digi>)}*