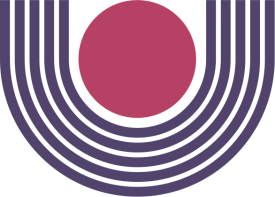
**UNIOESTE – Universidade Estadual do Oeste do Paraná**

Campus de Cascavel PR - Rua Universitária, 1619

**Bacharelado em Ciência da Computação - CCET**

Colegiado de Ciência da Computação

|  |  |  |
| --- | --- | --- |
| Disciplina: | **Compiladores** |  |
| Alunos: | **Juliano Felipe da Silva** |  |
| **Maycon de Queiroz Oliveira** | 06/08/2017 |

Descrição da linguagem de programação NOME

**N**on **O**bject Oriented and **M**ockingly **E**nvisaged language

# Operadores

## 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 |

## 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 |
|  |  |  |

## 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 |

# 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.

## 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 |

## INT

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

## FLT

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

# Estruturas de desvios

## Desvios Incondicionais

### BRK

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

## Desvios condicionais

### 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){

}

### 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 {

}

# Estruturas de repetição

## 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){

}

## 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){

}

# Estruturas de I/O

## Estruturas de Entrada de dados

Para entrada de dados, será utilizada a função ‘scn’, nativamente suportada pela linguagem, uma variável deve ser colocada em seguida entre parênteses para ser atribuída a entrada que será informada. Exemplo:

scn (A);

## Estruturas de Saída de dados

Para saída de dados, será utilizada a função ‘prt’, nativamente suportada pela linguagem, Um trecho de texto ou uma variável deve ser colocada em seguida entre parênteses (e aspas no caso de texto), os delimitadores \t (para tabulações), \n (para quebras de linhas) e \\ (para impressão da barra invertida) são aceitos em texto. Exemplo:

prt (“O valor de A e: “);

prt (A);

prt(“\n”);

# Definições

## Palavras reservadas

Todos os comandos são palavras reservadas como apresentado na tabela abaixo, os comandos apenas são palavras reservadas em letras minúsculas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| chr | int | flt | brk | scn |
| iff | els | for | whl | prt |

## 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.

## Comentários

Comentários de linha são permitidos e sempre devem ser precedidos por # (cerquilha). Exemplo:

#Comentário que não será levado em consideração

# Formalismos

## EBNF geral da linguagem

Todo o desenvolvimento de EBNF de um programa nessa linguagem se inicia com <begi>.

|  |  |  |
| --- | --- | --- |
| <begi> | → | <stmt> |
| <stmt> | → | <atri>[<stmt>] | <decl\_stmt>[<stmt>] | <rept\_stmt>[<stmt>] | <cond\_stmt>[<stmt>] | <ql>[<stmt>] | <ws>[<stmt>] | <cmnt>[<stmt>] |
| <cmnt> | → | #{(<simb> | <letr> | <digi> | <ws>)}\*<ql> |
| <oper\_logi> | → | (<id> | <letr> | <num>) (& | ‘|’ | ^ | ‘<<’ | ‘>>’) (<id> | <letr> | <num>) | !(<id> | <letr> | <num>) |
| <oper\_arit> | → | (<id> | <letr> | <num>) (+| - | \* | / | %) (<id> | <letr> | <num>) |
| <oper\_term> | → | <id> (& | ‘|’ | ^ | ‘<<’ | ‘>>’ | + | - | \* | / | %) ‘(‘<term>’)’ | ! ‘(‘<term>’)’ | ‘(‘<term>’)’ (& | ‘|’ | ^ | ‘<<’ | ‘>>’ | + | - | \* | / | %) <id> |
| <term> | → | <oper\_term> | <oper\_logi> | <oper\_arit> |
| <atri> | → | <id> = <term>; | <id> = (<id> | <letr> | <num>); |
| <decl\_stmt> | → | (chr | int | flt | bln) <id>; |
| <lexp> | → | <id> (== | != | ‘<’= | ‘>’= | ‘<’ | ‘>’) (<id> | <letr> | <num>) |
| <cond\_stmt> | → | iff’(‘<lexp>’)’’{‘<stmt><stmt>\*’}’ [els ‘{‘<stmt><stmt>\*’}’] | brk; |
| <rept\_stmt> | → | for’(‘<atri>; <lexp>; <atri>’)’’{‘<stmt><stmt>\*’}’ | whl’(‘<lexp>’)’’{‘<stmt><stmt>\*’}’ |
| <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 |
| <simb> | → | ! | “ | # | $ | % | & | ‘’’ | ‘(‘ | ‘)’ | \* | + | , | - | . | / | : | ; | ‘<’ | = | ‘>’ | ? | @ | ‘[‘ | ‘]’ | \_ | ^ | ´ | ` | ~ | ‘{‘ | ‘}’ | ‘|’ |
| <ql> | → | Quebra de linha |
| <ws> | → | Tabulação, espaçamento |
| <num> | → | <digi>{<digi>}\* | <digi>{<digi>}\*.<digi>{<digi>}\* |
| <id> | → | <letr>{<letr> | <digi>}0~254 |

## Expressão regular da linguagem

chr | int | flt | brk | iff | els | for | whl | #.\* | ! | & | ‘|’ | ‘+’ | - | ‘\*’ | / | ; | > | < | = | { | } | == | != | <= | >= | % | ^ | << | >> | ( | ) | “

## Autômato de reconhecimento de *tokens* da linguagem

