**PARADIGMAS DE PROGRAMAÇÃO**

**EXERCÍCIOS LISTA 05**: EXPRESSÕES E SENTENÇAS DE ATRIBUIÇÃO

Prof.ª Lucília Ribeiro

Aluno: **Thiago Santos Oliveira** Matrícula:20141002803464

**RESPOSTA 01:**

a) PRECEDÊNCIA DE OPERADOR: As regras de precedência de operadores para avaliação de expressões definem a ordem pela qual os operadores de diferentes níveis de precedência são avaliados. As regras de precedência de operadores para expressões são baseadas na hierarquia de prioridades dos operadores, conforme visto pelo projetista da linguagem. As regras de precedência de operadores das linguagens imperativas comuns são praticamente todas iguais, porque são baseadas naquelas da matemática. Nessas linguagens, a exponenciação tem a mais alta precedência (quando fornecida pela linguagem), seguida pela multiplicação e divisão no mesmo nível, depois pela adição e subtração binária no mesmo nível.

b) ASSOCIATIVIDADE DE OPERADOR: Quando uma expressão contém duas ocorrências adjacentes (separados por um único operando) de operadores com o mesmo nível de precedência, a questão sobra qual operador é avaliado primeiro é respondida pelas regras de associatividade da linguagem. Um operador pode ter associatividade à direita ou à esquerda, ou seja, que a ocorrência mais à esquerda é avaliada primeiro ou a ocorrência mais à direita é avaliada primeiro, respectivamente.

c) EFEITO COLATERAL FUNCIONAL: Ocorre quando a função modifica um de seus parâmetros ou uma variável global.

d) COERÇÃO: Definida como uma conversão de tipo implícita iniciada pelo compilador em expressões de modo misto.

e) OPERADOR SOBRECARREGADO: Uso múltiplo de um operador. Prática considerada aceitável, desde que nem legibilidade nem confiabilidade sejam comprometidas. Por exemplo, o operador + é usado para especificar a adição tanto de inteiros quanto de valores de ponto flutuante.

f) CONVERSÃO DE ALARGAMENTO: Converte um valor para um tipo que pode incluir ao menos aproximações de todos os valores do tipo original. Por exemplo, converter um int para um float em Java é uma conversão de alargamento.

CONVERSÃO DE ESTREITAMENTO: Converte um valor de um tipo que não pode armazenar aproximações equivalentes a todos os valores do tipo original. Por exemplo converter um double para um float.

g) CAST: Conversões de tipo explícitas solicitadas pelo programador são referidas como conversões explícitas (casts), não como coerções.

**RESPOSTA 02:** Avaliação de curto-circuito, avaliação mínima ou avaliação de McCarthy especifica a semântica de alguns [operadores booleanos](https://pt.wikipedia.org/w/index.php?title=Opera%C3%A7%C3%A3o_booleana&action=edit&redlink=1) em algumas [linguagens de programação](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o) na qual o segundo argumento é apenas executado ou avaliado se o primeiro argumento não for suficiente para determinar o valor da expressão: quando o primeiro argumento de uma função AND é avaliado como falso, o valor global deve ser falso e quando o primeiro argumento da função OR for avaliado como verdadeiro, o valor global deve ser verdadeiro. Por exemplo, o valor da expressão aritmética (13\*a)\*(b/13-1) é independente do valor de (b/13-1) se a for igual a 0.

**RESPOSTA 03:** A precedência dos operadores aritméticos, relacionais e booleanos nas linguagens baseadas em C é:

*Mais alta* ++ e –pós-fixados

+ e – unários, ++ e – pré-fixados, !

\*, /, %

+ e – binários

<, >, <=, >=

=, !=

&&

*Mais baixa* ||

Uma expressão legal em C é, a>b>c, onde o operador relacional mais à esquerda é avaliado primeiro porque os operadores relacionais em C são associativos à esquerda.

Quando os operadores não aritméticos das linguagens baseadas em C são incluídos, existem mais de 40 operadores e ao menos 14 níveis de precedência diferentes.

**RESPOSTA 04**:

a)(((a \* b )¹ - 1 )² + c )³

b)(((a\* (b – 1 )¹ )² / c )³ mod d )4

c)(((a - b )¹ / c )6 & ((((d \* e )³ / a )4 – 3 )5 )² )7

d)((- a)¹ or ((c = d )² and e )³ )4

e)(((a > b )¹ xor c )³ or (d <= 17 )² )4

f)((- a)¹ + b)²

**RESPOSTA 05:**

a)(a \* (b - (1 + c )¹ )² )³

c)(a \*(b - (1 / (c mod d )¹ )² )³ )4

d) (- (a or (c = (d and e )¹ )² )³ )4

e)(a >(b xor (c or (d <= 17 )¹ )² )³ )4

f)(- (a + b)¹ )²

**RESPOSTA 06:**

<expr> -> <expr> or <relação1>

|<expr> xor <relação1>

|<relação1>

<relação1> -> <relação1> and <relação2>

|<relação2>

<relação2> -> <relação2> = <termo1>

|<relação2> /= <termo1>

|<relação2> < <termo1>

|<relação2> <= <termo1>

|<relação2> >= <termo1>

|<relação2> > <termo1>

|<termo1>

<termo1> -> <termo1> - <unário>

|<unário>

<unário> -> <unário> + <termo>

|<unário> - <termo>

|<unário> mod <termo>

|<termo>

<termo> -> <termo> \* <fator>

|<termo> / <fator>

|<termo> not <fator>

|<fator>

<fator> -> (<expr>)

|<id>

<id> -> a|b|c|d|e

**RESPOSTA 07:**

**????????????????????????**

**RESPOSTA 08:**

Há efeito colateral na função fun, logo a ordem de avaliação dos operandos das expressões soma1 e soma2 altera os resultados, respectivamente, 46 e 48.

RESPOSTA 09:

**RESPOSTA 09:**

Em C++ os resultados são: Soma1= 46, Soma2=48

Já em Java são: Soma1=46 e Soma2=46;

Pois k foi passado por valor para a função fun em Java.

**RESPOSTA 10:**

#include <stdio.h>

int fun(){

return 3;

}

int main(){

int a, b;

a = 10;

b = a + fun();

printf("Com a chamada a função pela direita, ");

printf("b = %d\n", b);

a = 10;

b = fun(a) + a;

printf("Com a chamada a função pela esquerda, ");

printf("b = %d\n", b);

}

**EXERCÍCIO 11:**

//INTEIRO - Compilation Time: 2,25s

//PONTO FLUTUANTE - Compilation Time: 2,13s

#include <iostream>

using namespace std;

int main(){

int a=10,b=8384.3,c=234.9,d=423423.44,e=234234.2,f=2432.234234,g=234234.234234,h,i,j;

h=a+b+c;

i=b+c/a+f;

j=a\*(f+9.99);

a=b\*b+a+c+d-f/a\*j+(a/b)+c\*d;

b=b\*b+a+c+d-f/a\*j+(a/b)+c\*d;

c=b\*b+a+c+d-f/a\*j+(a/b)+c\*d;

system("pause");

return 0;

}