9,2
EXCELENTE!!

2ª Lista de exercícios de Paradigmas de Programação
Prof. Glauber Cintra – Entrega: 19/nov/2012

Alunos: Dênis Rainer, Lucas Diego, Cristiane Oliveira

Essa lista deve ser feita por grupos de no **mínimo** 3 e no **máximo** 4 alunos.

1) Qual o propósito dos *parênteses* em expressões? Como seria possível eliminá-los?

✓ 0,7 Os parênteses alteram as regras de precedência e associatividade. Convertendo as expressões infixas em pré-fixas e pós-fixas.

2) Defina o que é *precedência dos operadores* e *associatividade dos operadores*.

✓ 0,7 Precedência é a ordem de avaliação dos operadores. Associatividade define a ordem de avaliação quando temos operadores de mesma precedência.

3) O que é *atribuição múltipla*? Cite uma linguagem que possui tal recurso.

✓ 0,3 É a possibilidade de atribuir um valor a muitas variáveis em uma única instrução. A linguagem C possui tal recurso.

4) Cite exemplos de desvios condicionais, desvios incondicionais, laços condicionais e laços contados (um de cada).

Desvio condicional

```
if(idade > 21)
        printf("Você é de maior !!!\n");
else {
        printf("Você é de menor !!!\n");
}
```

✓ 0,8

Desvio incondicional

```
goto LABEL(local do código para onde se deve desviar)
```

Laços condicionais

```
while(n < k)
{
        //instruções a serem executadas
```

```
        n++;
}
```

Laços contados

```
for(n = 0; n < k ; n++)
{
        //instruções a serem executadas
}
```

5) Descreva detalhadamente o funcionamento da estrutura *switch* na linguagem Java.

A estrutura "switch" realiza a verificação de uma variável e age de acordo com o valor dessa variável. Cada valor esperado pela estrutura "switch" é denominado um caso ou "case", de acordo com o *case* (valor da variável), um trecho de código específico é executado. O switch tem a finalidade de controlar várias ações diferentes de acordo com o *case* definido e o respectivo trecho de código.

```
Exemplo: switch(variável) {
        case valor_1:
                        // código a ser executado para o caso do valor_1
                break;
        case valor_2:
                // código a ser executado para o caso valor_2
                break;
        .
        .
        .
        case valor_n:
                // código a ser executado para o caso valor_n
                break;
        default
                //código a ser executado caso nenhum dos valores esperados seja atendido

        }
```

1 0,6

Uma boa prática de programação é sempre terminar um código após o case por um comando break. Assim é possível evitar que o resto do código seja executado por acidente.

6) Explique o funcionamento do trecho de código abaixo. Para que serve tal código?

```
for (q = 0; n > = d; n = n – d) { q++; }
```

O trecho de código trabalha da seguinte forma:

Primeiramente a variável q é inicializada com 0 (zero), logo após é verificada a validade da condição, a variável n é maior ou igual a variável d. Caso afirmativo a execução segue incrementando a variável q de uma unidade e a variável n é atualizada recebendo o valor da subtração de n por d. Dá-se início a uma nova iteração onde a condição é verificada novamente. Caso a condição falhe o laço para .

O código implementa o algoritmo de divisão inteira de dois números naturais. ✓ 0,8

7) Explique como é constituída a assinatura de um método em Java.

Todo método definido tem uma assinatura que garante que não haja dois métodos iguais no programa. Assim é possível codificar em uma classe dois ou mais métodos com o mesmo nome, mas com assinaturas diferentes. Os elementos de um método que fazem parte de sua assinatura são:

> modificadores de acesso;
> tipo de retorno;
> nome do método;
> especificação de parâmetros;
> quantidade de parâmetros;
> tipo de cada parâmetro;
> ordem de seus parâmetros.

✓ 0,5

8) Explique os mecanismos de passagem de parâmetros por valor-resultado e por nome e indique que modelo semântico eles implementam.

Valor-Resultado: no início da execução do subprograma, o valor do parâmetro real é copiado para o parâmetro formal; no final da execução do subprograma, o valor do parâmetro formal é copiado para o parâmetro real. Este modelo semântico implementa a passagem de parâmetro da forma *in-out,* onde a passagem de dados ocorre tanto do chamador para o subprograma, quanto do subprograma para o chamador.

✓ 0,8

Por nome: o parâmetro real substitui todas as ocorrências do parâmetro formal dentro do subprograma. Este modelo também implementa a passagem de parâmetros da forma *in-out.*

9) Indique qual será o resultado produzido pelo programa:

```
pai(rui, gil).
pai(gil, ivo).
pai(gil, edu).
avo(X, Y) :- pai(X, Z), pai(Z, Y).
```

para as seguintes metas:

pai(rui, edu). > **false**

pai(gil, edu). > **true**
pai(R, edu). > **R = gil**
pai(gil, R). > **R = ivo**
avo(rui, gil). > **false**
avo(rui, edu). > **true**
avo(R, gil). > **false**
avo(rui, R). > **R = ivo**

✓ 0,8

10) Defina um predicado em Prolog que receba um número inteiro (positivo ou negativo) e escreva 2*n*. Por exemplo, se o seu predicado se chamar potencia, a meta potencia(-2) deverá escrever 0.25.

pow(0,1).
pow(1,2).
pow(P,R):- P > 0, A is P-1, **pow**(A,B), R is 2*B.
pow(P,R):- P < 0, Z is -P,**pow**(Z,C), R is 1/C.
pow(P):- **pow**(P,R), write(R).

✓ 0,8

11) Defina um predicado em Prolog que receba uma lista de números e escreva a multiplicação dos elementos da lista. Por exemplo, se seu predicado for chamado de mult, a meta mult([3, 2, 5]) fará com que o valor 30 seja escrito.

mult([], 1).
mult([W],W).
mult([Cb|Cd], M):- **mult**(Cd, B), M is Cb*B.
mult([Cb|Cd]):- **mult**([Cb|Cd], R), write(R).

✗ 0,7

12) Descreva a semântica da função COND.

✓ 0,6

A função COND em Lisp implementa o desvio condicional, tendo como parâmetros pares do tipo condição-ação, essas condições são avaliadas em sequencia e quando uma delas devolve um valor diferente de *nil* ,sua ação é avaliada e seu resultado é passado como retorno da função COND, o restante das condições não é avaliada. Caso todas as condições forem avaliadas como *nil*, a função retorna *nil.*

13) Escreva uma função em Lisp que receba um número inteiro positivo e devolva a soma dos seus divisores positivos (você pode definir mais de uma função para resolver essa questão).

```
(defun mult (n y L) (if (= n 0)
        L
        (cond ((= n y) (append (list y) L))
```

```
                ( (if (= (mod n y) 0)
                      t
                      nil)
                      (mult n (+ y 1) (append (list y) L) )
                      )
                      (t  (mult n (+ y 1)  L ) )
        )
    )
)

(defun multiplo (n) (mult n 1 () ) )

(defun soma(L)(if (null L)
        0
        (+ (car L) (soma (cdr L))))
    )

(defun somamultiplo (n) (soma (multiplo n)))
```

14) Escreva uma função em Lisp que receba uma lista de números e devolva *t* se ela estiver em ordem crescente; *nil*, caso contrário.

```
(defun desordem (n L)( if (null L)
        nil
        (if (> n (car L))
                t
                (desordem (car L) (cdr L))
        )
    )
)

(defun ordenado (L) (if (desordem (car L) (cdr L))
                        nil
                        t
        )
)
```