CONSISTÊNCIA DOS DADOS DE ENTRADA

CONSISTÊNCIA E MODULARIZAÇÃO

“Consistir os dados” significa validá-los, ou seja, verificar se os valores digitados como entrada são válidos ou não. Para isso, trabalhamos com as “críticas” ou mensagens de alerta, que serão exibidas ao usuário, referentes aos valores digitados que não atendam às condições estabelecidas. Por exemplo, quando desejamos calcular a nota de um aluno por média aritmética, solicitamos a entrada de 3 números válidos entre 0 e 10. Logo, o intervalo de valores permitidos varia de 0 a 10, não sendo aceitos números negativos e nem maiores do que 10. No caso de digitação errada ou proposital por parte do usuário, o algoritmo exibe uma mensagem de crítica, alertando-o para que digite de forma correta os valores solicitados.

MODULARIZAÇÃO

Existem algoritmos de todas as complexidades, sendo que esta característica está diretamente relacionada com a finalidade que o algoritmo apresenta, com o tipo de negócio ao qual está associado, bem como os requisitos e regras de negócios para a elaboração do mesmo. Algoritmos que apresentam muitas regras em sua construção tendem a ficar com código extenso, dificultando a sua interpretação e futura manutenção por trechos de código que não ficam claros ou que são repetidos dentro da lógica de construção.

Uma solução eficaz para o problema descrito é a modularização, ou seja, um algoritmo maior é quebrado em módulos, ou subalgoritmos. Esta técnica facilita a compreensão, além de organizar melhor o código, permitindo a sua reutilização. Na prática, dividimos um algoritmo maior em diversas partes menores – tantas quantas forem necessárias. O módulo principal se diferencia dos outros por ser o início da execução. Este módulo é que faz a chamada e contempla em sua estrutura a chamada dos módulos menores. Importante ter em mente que a chamada de um módulo significa a execução das regras contidas nele. Terminada a execução do módulo, esta volta para o ponto do módulo, podendo seguir com a programação estabelecida no módulo principal, chamar outro módulo ou até mesmo o módulo que acabou de ser executado, caso o mesmo se encontre dentro de uma condição de repetição. Um módulo nada mais é do que um grupo de comandos que constitui um trecho de algoritmo com uma função bem definida

o mais independente possível das demais partes do algoritmo. Cada módulo, durante a execução do algoritmo, realiza uma tarefa específica da solução do problema e, para tal, pode contar com o auxílio de outros módulos do algoritmo. Desta forma, a execução de um algoritmo contendo vários módulos pode ser vista como um processo cooperativo. A construção de algoritmos compostos por módulos, ou seja, a construção de algoritmos através de modularização possui uma série de vantagens:

• Torna o algoritmo mais fácil de escrever. O desenvolvedor pode focalizar em pequenas partes de um problema complicado e escrever a solução para estas partes, uma de cada vez, ao invés de tentar resolver o problema como um todo de uma só vez.

• Torna o algoritmo mais fácil de ler. O fato do algoritmo estar dividido em módulos permite que alguém, que não seja o seu autor, possa entender o algoritmo mais rapidamente por tentar entender os seus módulos separadamente, pois como cada módulo é menor e mais simples do que o algoritmo monolítico correspondente, entender cada um deles separadamente é menos complicado do que tentar entender o algoritmo como um todo de uma só vez.

• Eleva o nível de abstração. É possível entender o que um algoritmo faz por saber apenas o que os seus módulos fazem, sem que haja a necessidade de entender os detalhes internos aos módulos. Além disso, se os detalhes nos interessam, sabemos exatamente onde examiná-los.

• Economia de tempo, espaço e esforço. O módulo principal solicita a execução dos vários módulos em uma dada ordem, os quais, antes de iniciar a execução, recebem dados do módulo principal e, ao final da execução, devolvem o resultado de suas computações.

COMPONENTES DE UM MÓDULO

Os módulos que estudaremos daqui em diante possuem dois componentes: corpo e interface. O corpo de um módulo é o grupo de comandos que compõe o trecho de algoritmo correspondente ao módulo. Já a interface de um módulo pode ser vista como a descrição dos dados de entrada e de saída do módulo. O conhecimento da interface de um módulo é tudo o que é necessário para a utilização correta do módulo em um algoritmo. A interface de um módulo é definida em termos de parâmetros. Um parâmetro é um tipo especial de variável utilizado pelos módulos para receber ou comunicar valores de dados a outras partes do algoritmo. Existem três categorias de parâmetros:

• parâmetros de entrada, que permitem que valores sejam passados para um módulo a partir do algoritmo que solicitou sua execução;

• parâmetros de saída, que permitem que valores sejam passados do módulo para o algoritmo que solicitou sua execução;

• parâmetros de entrada e saída, que permitem tanto a passagem de valores para o módulo quanto a passagem de valores do módulo para o algoritmo que solicitou sua execução. Quando criamos um módulo, especificamos o número e os tipos das variáveis correspondentes aos parâmetros que ele necessita, isto determina a interface do módulo. Qualquer algoritmo que use um módulo deve utilizar os parâmetros que são especificados na interface do módulo para se comunicar com ele.

Exemplo

```
início
        {declarações globais}
        módulo principal;
        {declarações locais}
        início
        {corpo do algoritmo principal}
fim;

        {definições de outros módulos ou subalgoritmos}
fim.
```

- Declarações globais: são variáveis, tipos e/ou constantes declaradas no início do algoritmo que podem ser utilizadas por qualquer módulo, inclusive pelo módulo principal.
- Módulo principal: é por onde se inicia a execução do algoritmo.
- Declarações locais: são variáveis, tipos e/ou constantes que só podem ser utilizadas dentro do módulo no qual foram declaradas.
- Corpo do algoritmo: conjunto de ações ou comandos.
- Definição dos módulos ou subalgoritmos: compreende um cabeçalho chamado módulo seguido de um nome, declarações locais e o corpo do subalgoritmo.

Exemplo de Declaração de um módulo:

```
módulo identificador;
                    {declarações locais}
                        início
                            {corpo do subalgoritmo}
                        fim;
```

EXEMPLO DE ALGORITMO COM MÓDULOS

Considere um aluno com três notas. Com base nisso, calcularemos a média deste aluno e exibiremos sua situação de acordo com a média obtida.

```
Algoritmo media
            Nome     texto;
            Nota1    numerico;
          Nota2      numerico;
          Nota3      numerico;
            Situacao  texto;
Início
   Início

       Leia(nome);
       Leia(nota1);
       Leia(nota2);
       Leia(nota3);

      --Chamada do módulo de cálculo
 calculo;
   Fim;
```

Em linguagens de programação, os módulos ou subalgoritmos, são chamados de procedimentos ou funções.

FUNÇÕES E PROCEDIMENTOS EM PORTUGOL

São duas as formas de definição de módulos em Portugol e nas linguagens de programação.

• Função: uma função é um módulo que produz um único valor de saída. Ela pode ser vista como uma expressão que é avaliada para um único valor, sua saída, assim como uma função em Matemática.

• Procedimento: um procedimento é um tipo de módulo usado para várias tarefas, não produzindo valores de saída. As diferenças entre as definições de função e procedimento permitem determinar se um módulo para uma dada tarefa deve ser implementado como uma função ou procedimento. Nas definições acima, estamos considerando que a produção de um valor de saída por uma função é diferente da utilização de parâmetros de saída ou de entrada e saída. Veremos mais adiante que os parâmetros de saída e de entrada e saída modificam o valor de uma variável do algoritmo que a chamou, diferentemente do valor de saída produzido por uma função que será um valor a ser usado em uma atribuição ou envolvido em alguma expressão. Como exemplo da diferenciação do uso de funções ou procedimentos, considere, por exemplo, que um módulo para determinar o menor de dois números é necessário. Neste caso, o módulo deve ser implementado como uma função, pois ele vai produzir um valor de saída. Por outro lado, se um módulo, para determinar o maior e o menor valor de uma sequência de números, é requerido, ele deverá produzir seus resultados via os parâmetros de saída e será implementado como um procedimento, pois ele vai produzir dois valores de saída. A fim de escrevermos uma função ou procedimento, precisamos construir as seguintes partes: interface e corpo.

A forma mais simples de resolver problemas através de algoritmos consiste em elaborar construções com apenas quatro tipos de estruturas de controle de fluxo: comandos de entrada e saída, comandos de atribuição e repetição, e, comandos condicionais. São usadas ainda as expressões lógicas e aritméticas. Além disso, precisamos nos preocupar com as limitações impostas pelos compiladores de cada ferramenta no momento de elaborar a lógica a ser executada. Várias são as situações que contribuem para o entendimento e adequada manutenção dos programas. Por isso, a construção deve sempre considerar formas de facilitar o trabalho dos analistas e programadores. Isso é possível elaborando-se o código em partes bem definidas e que contenham partes importantes e coerentes do problema em si. Ter conhecimento e domínio da construção de procedimentos e funções é o melhor caminho para a construção de códigos de programação com qualidade, facilitando a manutenção e reutilização, sempre que necessário. Procedimentos e funções são subprogramas, ou seja, são pedações de um programa menor, dentro de um programa maior. Este particionamento permite a construção de programas mais enxutos, porém, altamente elaborados, pois trabalham de forma modular, com reaproveitamento e maior legibilidade do código.

PASSAGEM DE PARÂMETROS

Quando construímos algoritmos de forma modularizada é muito comum precisar passar informações de um módulo para outro. Essas informações são denominadas parâmetros. Os parâmetros são responsáveis por estabelecer a comunicação entre os módulos. Um parâmetro que sai de um módulo é um valor de entrada para o módulo que está sendo chamado, sendo que a lógica do módulo é que vai definir a forma de manipulação da informação: se será somente para leitura ou se o valor será alterado. Caso esta informação seja alterada no novo módulo, isto não significa que é alterada no módulo de origem, onde permanece inalterada. Um parâmetro também pode ser chamado de argumento. Existem dois tipos distintos de parâmetros: Passagem de parâmetros por valor: Consiste em copiar o valor das variáveis locais usadas na lógica e passá-las para outro módulo sem alterar informações originais. Passagem de parâmetros por referência: Neste caso, é feita a cópia do endereço da memória onde a variável está armazenada. Nesse mecanismo, outra variável ocupando outro espaço diferente na memória não armazena o dado em si, e sim o endereço onde ele se localiza na memória. Assim, todas as modificações efetuadas nos dados do parâmetro estarão sendo feitas no conteúdo original da variável. Podemos identificar os dois tipos de situação em um mesmo algoritmo.

EXEMPLO 2: Dados dois valores positivos, calcular e exibir o resultado das operações aritméticas efetuadas utilizando passagem por parâmetros.

```
início

    módulo principal;
  real: x, y, res;   {variáveis locais}
    início
      leia(x);
      leia(y);
      se (x>0) e (y>0) então
            início
              calc(x,y,'+', res);
   escreva(res);
              calc(x,y,'-', res);
        escreva(res);
              calc(x,y,'*', res);
              escreva(res);
              calc(x,y,'/', res);
              escreva(res);
            fim;
  fim;
    módulo calc(real:a,b;caracter:oper; ref real:re);


        início
          escolha oper
                caso '+' : re ← a + b;
                caso '-' : re ← a - b;
                caso '*' : re ← a * b;
                caso '/' : re ← a / b;
          fim;
        fim;
fim.
```

Ainda podemos ter um módulo que tenha a função de calcular um resultado, como exemplo as funções matemáticas. Este tipo de módulo tem o objetivo de retornar uma única informação ao módulo chamador, como se fosse uma resposta. Esta resposta será feita através do comando retorne(): Declaração: retorne(expressão); onde: expressão: pode ser uma informação numérica, alfanumérica ou uma expressão aritmética.

EXEMPLO 3: O mesmo d o exemplo 1 modificado.

```
início
        módulo principal;

início
  entrada;
fim;

módulo entrada;
real:re;
caracter:op;
início
        repita
                leia(re);
                escreva(conv(re));


                        leia(op);
              até op='N';
    fim;

    módulo conv(real:r);
    início
                retorne(r/1.20);
    fim;
fim.
```

EXEMPLO 4: O mesmo do exemplo 2 modificado usando retorne. Observe que sempre irá retornar um único resultado de cada vez.

```
início
        módulo principal;
      real: x, y;   {variáveis locais}
      início
        leia(x);
        leia(y);
        se (x>0) e (y>0) então
                início
                  escreva(calc(x,y,'+'));
                  escreva(calc(x,y,'-'));
                  escreva(calc(x,y,'*'));
                  escreva(calc(x,y,'/'));
                fim;
      fim;
        módulo calc(real:a,b;caracter:oper);
        início
          escolha oper
                caso '+' : retorne(a + b);
                caso '-' : retorne(a – b);
                caso '*' : retorne(a * b);
```

```
                caso '/' : retorne (a / b);
            fim;
        fim;
fim.
```

Antes de se definir pela modularização é importante pensar nas seguintes questões:

• Saber exatamente quando usar procedure e quando usar uma função.

• Saber a diferença entre variáveis locais e globais e quando usá-las.

• Saber quando qual tipo de passagem de parâmetros deverá ser usado: valor ou referência.

Imagine um algoritmo que você precisa ler uma sequência de valores (números inteiros terminados com zero) exibindo como resultado somente os números pares. Exemplo:

```
Algoritmo Pares;;

      Num   inteiro;
Inicio
   Leia (num);
     Enquanto num <> o faça
        Se num mod2 = 0 entao
                  Escreva ('Num');
                   Leia (Num)
              Fim-se;
          Fim enquanto;
Fim;
```

Note que no exemplo acima o código inteiro está no programa principal, entre o comando "inicio" e o comando "Fim".

VARIÁVEIS GLOBAIS

São as variáveis declaradas logo após o cabeçalho do programa (seu título) e antes do comando de início (Inicio) da execução do programa principal. As variáveis são endereços de memória (local) onde os dados ficam armazenados temporariamente até serem usados. As variáveis globais podem ser utilizadas dentro do programa onde foram criadas, mas não fora dele. No exemplo anterior, temos uma única variável. Seu nome é Num e seu tipo é inteiro.

VARIÁVEIS LOCAIS

As variáveis locais são declaradas dentro dos subprogramas e só podem ser usadas dentro dos mesmos. Elas não podem ser usadas pelo programa principal.

FUNÇÕES

No exemplo anterior (do algoritmo que verifica se um número é par), como poderíamos usar uma Função?

Note que existe um cálculo no programa principal, representado pelo código Num mod2 = 0. Este cálculo é uma expressão booleana (retorna verdadeiro ou falso) que calcula a divisão de um número por 2 e verifica se o resultado é zero. Se for zero, significa que o número é par. Caso contrário, não. O exemplo proposto é simples, apresentando poucas linhas de código em sua construção, mas imagine um cálculo mais complexo, com muitas linhas de código e muitas checagens até se chegar ao resultado final. Esta parte mais extensa e bem específica (do cálculo) poderia ser transformada em uma função. Basicamente, o programa principal passaria como parâmetro para a função um número lido e se encarregaria de validar as checagens, devolvendo o resultado. Esta forma de escrita cria um vínculo entre o programa principal e o subprograma. Toda função apresenta as seguintes características:

• Toda função tem um nome.

• Toda função pode ou não receber parâmetros ou argumentos de entrada.

• Toda função retorna, obrigatoriamente, um valor de um único tipo de dado (data, texto ou número).

Vejamos o exemplo:

```
Algoritmo função 01;
program imprime pares;

        Num   inteiro:

( funcao que verifica se o número recebido como parâmetro é par )

    Funcao  verifica par  (Num inteiro)  retorna boolean
      Epar   boolean;

inicio
se Num mod 2 = 0 then
Epar := verdadeiro;
senao
Epar := falso;
            Fim se;
      Fim;

Inicio  (programa principal)


Leia (Num) ;
Enquanto Num <> 0 faça

   Se Epar ( Num) então –(aqui ocorre a passagem de parâmetro por valor)
Escreva (Num);
Leia (Num);
                Fim-se;
      Fim;
```

Perceba que transferir parte do código para um sobprograma não altera e nem interfere na execução do programa principal. É importante notar que o código fica mais legível e organizado.

PROCEDIMENTOS

Os procedimentos se diferem das funções no sentido de que podem não retornar nenhum valor, ou mais de um inclusive com tipos de dados distintos. Na prática, um procedimento poderia retornar um valor texto, um valor numérico e uma data, por exemplo. A sintaxe de criação da procedure se resume em informar seu nome e a lista de parâmetros de entrada. O procedimento também precisa ter um nome definido. E caso existam, deve ocorrer também a passagem de parâmetros como entrada para a lógica que será desenvolvida. Exemplo de procedimento:

• Criar um procedimento que receba dois valores inteiros por parâmetro e realize a troca desses valores.

```
algoritmo "TrocarValores"
var
   x, y: inteiro

   procedimento troca(var a, b: inteiro)
   var
      auxiliar: inteiro
   inicio
      auxiliar <- a
      a <- b
      b <- auxiliar
   fimprocedimento

inicio
   x <- 5
   y <- 8
   escreval("Os valores de x e y ANTES da troca são: ", x, y)
   troca(x, y)
   escreval("Os valores de x e y DEPOIS da troca são: ", x, y)
fimalgoritmo
```

O nosso próximo assunto é vetores e matrizes.

Saiba mais

Leia sobre vetores e matrizes em: https://dicasdeprogramacao.com.br/o-que-sao-vetores-e-matrizes-arrays/

Caro Aluno: você, além de aprender a estruturar algoritmos em forma de vetores e matrizes, também estará apto a identificar as situações de uso destas estruturas.

CONCEITOS DE VETORES E MATRIZES

Vetores e matrizes são estruturas de dados muito simples que podem nos ajudar muito quando temos muitas variáveis do mesmo tipo em um algoritmo. Imagine o seguinte problema: Você precisa criar um algoritmo que lê o nome e as 4 notas de 50 alunos, tendo que calcular a média de cada aluno e informar quais foram aprovados e quais foram reprovados. Conseguiu imaginar quantas variáveis você vai precisar? Muitas né?! Vamos fazer uma conta rápida: 50 variáveis para armazenar os nomes dos alunos, (4 * 50 =) 200 variáveis para armazenar as 4 notas de cada aluno e por fim, 50 variáveis para armazenar as médias de cada aluno. 300 variáveis no total, sem contar a quantidade de linhas de código que você vai precisar para ler todos os dados do usuário, calcular as médias e apresentar os resultados. Mas temos uma boa notícia para você. Nós não precisamos criar 300 variáveis! Podemos utilizar Vetores e Matrizes (também conhecidos como ARRAYs)! Vetor (array unidimensional) é uma variável que armazena várias variáveis do mesmo tipo. No problema apresentado anteriormente, nós podemos utilizar um vetor de 50 posições para armazenar os nomes dos 50 alunos. Matriz (array multidimensional) é um vetor de vetores. No nosso problema, imagine uma matriz para armazenar as 4 notas de cada um dos 50 alunos, ou seja, um vetor de 50 posições, e em cada posição do vetor há outro vetor com 4 posições. Isso é uma matriz.

Cada item do vetor (ou matriz) é acessado por um número chamado de índice. Vamos representar os vetores e matrizes graficamente para facilitar o entendimento do conceito.

Perceba, na figura acima, que cada posição do vetor é identificada por um número (chamado de índice). No caso da matriz são dois números (um na vertical e um na horizontal). Vamos entender melhor desenvolvendo um algoritmo na prática. O exemplo descrito calcula a média de cinco alunos. Os dados são armazenados dentro de vetores para a exibição do resultado final. Importante perceber que nesta situação não é

possível a utilização de variáveis normais, as quais estudamos nos primeiros algoritmos, devido a sua incapacidade de armazenar mais de um valor.

```
algoritmo "MediaDe5Alunos"

var

   nomes: vetor [1..5] texto
   notas: vetor [1..5,1..4] numerico
   medias: vetor [1..5] de real
   contadorLoop1, contadorLoop2: inteiro

inicio

     //Leitura dos nomes e as notas de cada aluno
     PARA contadorLoop1 DE 1 ATE 5 FACA
          ESCREVA("Digite o nome do aluno(a) número ", contadorLoop1, " de 5: ")
          LEIA(nomes[contadorLoop1])
          PARA contadorLoop2 DE 1 ATE 4 FACA
               ESCREVA("Digite a nota ", contadorLoop2, " do aluno(a) ", nomes[contadorLoop1], ": ")
               LEIA(notas[contadorLoop1, contadorLoop2])
          FIMPARA
          //CÁLCULO DAS MÉDIAS
          medias[contadorLoop1] := (notas[contadorLoop1, 1] + notas[contadorLoop1, 2]
                                                            + notas[contadorLoop1, 3]
                                                            + notas[contadorLoop1, 4]) / 4
     FIMPARA

     //APRESENTAÇÃO DOS RESULTADOS
     PARA contadorLoop1 DE 1 ATE 5 FACA
          SE medias[contadorLoop1] >= 6 ENTAO
             ESCREVAL("O aluno(a) ", nomes[contadorLoop1], " foi aprovado com as notas (", notas[contadorLoop1, 1], ",
                                    ", notas[contadorLoop1, 2], ",
                                    ", notas[contadorLoop1, 3], ",
                                    ", notas[contadorLoop1, 4], ") e média: ", medias[contadorLoop1])
          SENAO
             ESCREVAL("O aluno(a) ", nomes[contadorLoop1], " foi reprovado com as notas (", notas[contadorLoop1, 1], ",
                                    ", notas[contadorLoop1, 2], ",
                                    ", notas[contadorLoop1, 3], ",
                                    ", notas[contadorLoop1, 4], ") e média: ", medias[contadorLoop1])
          FIMSE
     FIMPARA

fimalgoritmo
```

Podemos dizer que as matrizes e vetores são estruturas de dados que se organizam a partir de dados primitivos que já existem. Estas estruturas podem armazenar um conjunto de dados e são definidas como variáveis compostas. Estas variáveis compostas são classificadas de duas formas distintas. Variáveis homogêneas e heterogêneas. Variáveis compostas homogêneas são variáveis que armazenam vários elementos do mesmo tipo primitivo. Exemplo: um conjunto de números inteiros e positivos. As variáveis compostas heterogêneas são estruturas de dados (armazenam um conjunto de dados). Esses dados podem ser armazenados em dois tipos de variáveis: as variáveis unidimensionais (conhecidas como vetores) e as variáveis multidimensionais (conhecidas como matrizes).

Variáveis compostas unidimensionais – VETORES Precisam de um índice para individualizar um elemento do conjunto.

Declaração: tipo: identificador[n]; onde: identificador – é o nome de uma variável; n – é o tamanho do vetor

Exemplo: Média de 40 alunos Estrutura que se forma para guardar a nota e a posição:

8.0	7.0	5.5	9.5	6.4	9.9	1.0	.....	4.8

Para acessar cada elemento da variável composta unidimensional (vetor) é necessário especificar o nome da variável seguido do número do elemento que se deseja acessar (deverá estar entre colchetes). Este número tem o nome de índice. O índice pode variar de 0 até n -1. Por exemplo, se um vetor possui 40 valores, os índices válidos são entre 0 até 39.

Leia (media[0]);

media [2]

8.5; escreva(media[3]);

Exemplo 1: Dados 5 valores numéricos, elaborar um algoritmo que calcule e exiba a média dos valores dados e quantos desses valores estão acima da média. Resolução sem utilizar vetor: Algoritmo sem vetor inteiro:

```
Resolução sem utilizar vetor:
Algoritmo sem vetor

   inteiro:n1,n2,n3,n4,n5,cont;
  real:media; -- não é um número inteiro  -- pode apresentar casas decimais
   cont← 0;

início
      leia (n1);
      leia (n2);
      leia (n3);
      leia (n4);
      leia (n5);
         media :=  (n1+n2+n3+n4+n5)/5;
         escreva(media);

         Se n1 > media então
            Cont := cont +1;
         Senão se n2 > media
            Cont := cont +1;
         Senão se n3 > media
            Cont := cont +1;
```

```
    Senão se n4 > media
        Cont := cont +1;

    Senão se n5 > media
        Cont := cont +1;
    Fim-se;
    Escreva (cont);
Fim;
```

Exercícios

1 – Proposições podem ser compreendidas por um conjunto de palavras que servem para formar uma ideia que possui como resultado um valor lógico. Conectivos são operadores que podem ser representados por símbolos, e unem duas ou mais proposições para formar uma nova proposição composta. Existem conectivos de negação, conjunção, disjunção, disjunção exclusiva, condicional e bicondicional. Para determinar se uma proposição composta é verdadeira ou falsa, dependemos do valor lógico das proposições e do tipo de conectivo que as une. Utilizando os valores de entrada da tabela a seguir, desenvolva a tabela-verdade para a proposição composta "~(p ou q)", que também pode ser escrita como "~(p v q)".

p	q
V	V
V	F
F	V
F	F

Resposta:

* Inicialmente pode ser construída a tabela abaixo, que apresenta a coluna da disjunção (p ou q).

p	q	p V q
V	V	V
V	F	V
F	V	V
F	F	F

* Por fim, o resultado, onde deve ser feita a negação da coluna disjunção acima. Na tabela abaixo, a terceira coluna apresenta a resposta.

p	q	p V q	~(p V q)
V	V	V	**F**
V	F	V	**F**
F	V	V	**F**
F	F	F	**V**

2 – Proposição pode ser definida como uma sentença declarativa que pode ter seu valor verdadeiro ou falso, podendo ser simples ou composta. Uma proposição composta é a união de duas proposições simples ligadas por conectivo. Existem vários tipos de conectivos, como o de negação, conjunção, disjunção, disjunção exclusiva, condicional e bicondicional. A conjunção de duas proposições "p" e "q" ocorre quando os valores lógicos de ambas as proposições são verdadeiros. Utilizando os valores de entrada da tabela a seguir, desenvolva a tabela-verdade para a proposição composta "~p e ~q", que também pode ser representada por "~p ^ q".

p	**q**
V	V
V	F
F	V
F	F

Resposta:

* Inicialmente pode ser construída a tabela abaixo, que apresenta a negação para as proposições p e q.

p	q	~p	~q
V	V	F	F
V	F	F	V
F	V	V	F
F	F	V	V

* Por fim, o resultado, onde deve ser feita a conjunção para as colunas de negação das proposições p e q acima. Na tabela abaixo, a quarta coluna apresenta a resposta para a questão.

p	q	~p	~q	~p ^ ~q
V	V	F	F	**F**
V	F	F	V	**F**
F	V	V	F	**F**
F	F	V	V	**V**

3 – Uma tabela-verdade pode ser definida por uma tabela em que são dispostos os valores lógicos de proposições que estão sendo analisados. Uma proposição pode ser definida como uma sentença declarativa, que pode ter seu valor verdadeiro ou falso, podendo ser simples ou composta. A quantidade de linhas de uma tabela-verdade é determinada pelo cálculo de 2 elevado a n, sendo n a quantidade de proposições. Nesse sentido, se uma tabela-verdade tiver duas proposições, então a tabela-verdade terá quatro linhas. Existem vários tipos de conectivos, como o de negação, conjunção, disjunção, disjunção exclusiva, condicional e bicondicional. Chama-se disjunção de duas proposições, "p" e "q", quando o valor lógico de ao menos uma delas é verdadeiro. Utilizando os valores de entrada da tabela a seguir, desenvolva a tabela-verdade para a proposição composta "~(p ou ~q)", que também pode ser representada por "~(p v ~q)".

p	**q**
V	V
V	F
F	V
F	F

Resposta:

* Inicialmente pode ser construída a tabela abaixo, que apresenta a negação para a proposição q.

p	q	~q
V	V	F
V	F	V
F	V	F
F	F	V

* Em seguida, pode-se construir a seguinte tabela verdade para apresentar o valor lógico da proposição "(p v ~q)".

p	q	~q	(p v ~q)
V	V	F	V
V	F	V	V
F	V	F	F
F	F	V	V

* Finalmente, o resultado, onde deve ser feita a negação da disjunção acima (quarta coluna da tabela acima). Na tabela abaixo, a quinta coluna apresenta a resposta para a questão.

p	q	~q	(p v ~q)	~(p v ~q)
V	V	F	V	**F**
V	F	V	V	**F**
F	V	F	F	**V**
F	F	V	V	**F**

4 – Considere uma situação em que um professor que queira saber se existem alunos cursando, ao mesmo tempo, as disciplinas A e B, tenha implementado um programa que:

1) inicializa um array a de 30 posições que contém as matrículas dos alunos da disciplina A.
2) inicializa outro array b de 40 posições, que contém as matrículas dos alunos da disciplina B.

3) imprime a matrícula dos alunos que estão cursando as disciplinas A e B ao mesmo tempo.

Considere, ainda, que os arrays foram declarados e inicializados, não estão necessariamente ordenados, e seus índices variam entre 0 e n - 1, sendo n o tamanho do array.

1. para (i de 0 até 29) faça
2. para (j de 0 até 39) faça
3.

4.
5.
6. fim-para
7. fim-para

Com base nessas informações, assinale a alternativa CORRETA que apresenta o trecho a ser incluído nas linhas 3, 4 e 5 do código, para que o programa funcione corretamente:

3. se (a[i] = b[j]) entao
( ) 4. escreva (a[j])
5. fim-se
3. se (a[i] = b[j]) entao
(**x**) 4. escreva (a[i])
5. fim-se
3. se (a[i] = b[i]) entao
( ) 4. escreva (a[i])
5. fim-se
3. se (a[j] = b[i]) entao
( ) 4. escreva (a[j])
5. fim-se

5 – Os conectivos lógicos possuem a finalidade de ligar duas ou mais proposições, que são presentes nas proposições compostas. Duas checagens são importantes para determinar se as proposições compostas são verdadeiras ou falsas. A primeira, verificar o valor das proposições que compõem as sentenças, e a segunda, o tipo de conectivo que liga as proposições de uma mesma sentença.

Sobre a negação, analise as seguintes afirmativas:

I- A negativa de "O almoço é barato" é "Não vou almoçar".
II- Um sinal de til pode ser utilizado para representar a negação de uma proposição.
III- Na lógica proposicional, é possível negar uma proposição composta.
IV- A negação da proposição p é representada por p~.

Assinale a alternativa CORRETA:
( ) Somente a afirmativa IV está correta.
(**x**) As afirmativas II e III estão corretas.
( ) As afirmativas I, II e IV estão corretas.
( ) Somente a afirmativa I está correta.

6 – Os conectivos lógicos possuem a finalidade de ligar duas ou mais proposições, que são presentes nas proposições compostas. Duas checagens são importantes para determinar se as proposições compostas são verdadeiras ou falsas. A primeira, verificar o valor das proposições que compõem as sentenças, e a segunda, verificar o tipo de conectivo que liga as proposições de uma mesma sentença.

Sobre a disjunção, analise as seguintes afirmativas:

I- "Se Brasil está na América, então Brasil é ocidental" é um exemplo de disjunção.
II- Pode ser representada pelo caractere: v
III- As proposições são unidas pelo conectivo e.
IV- "Verde combina com branco ou amarelo combina com azul" é um exemplo de disjunção.

Assinale a alternativa CORRETA:
( ) Somente a afirmativa IV está correta.
( ) As afirmativas II e III estão corretas.
(**x**) As afirmativas II e IV estão corretas.
( ) Somente a afirmativa I está correta.

7 – Uma tabela verdade é utilizada para determinar o valor lógico de uma proposição composta, pois os valores de proposições simples já são conhecidos. O valor lógico de uma proposição composta depende do valor lógico da proposição simples que a constitui. É possível desenvolver uma tabela verdade para qualquer expressão lógica. Uma proposição composta é constituída de duas ou mais proposições simples ligadas por um conectivo. Para determinar a quantidade de linhas que uma tabela verdade terá, basta calcular 2 elevado à quantidade de proposições simples. Além disso, se duas proposições compostas possuem os mesmos valores verdade, elas são equivalentes. Dependendo dos valores lógicos de proposições compostas, pode ser dito que ocorre uma tautologia, contradição ou contingência. Neste contexto, analise a tabela verdade a seguir, comente se ocorre uma tautologia, contradição ou contingência, e explique o porquê.

p	q	$p \wedge q$	$p \rightarrow (p \wedge q)$
V	V	V	V
V	F	F	F
F	V	F	V
F	F	F	V

Resposta:

Sempre que uma proposição composta não for uma tautologia ou uma contradição, será uma contingência. Neste sentido, para ser uma contingência, é necessário que na última coluna da tabela verdade, exista ao menos uma linha com valor falso e uma com o valor verdadeiro, que é o que acontece com a tabela verdade anexa.

8 – Para um aprendizado concreto sobre algoritmos, é necessário conhecimento sobre fundamentos da lógica matemática. Com esse objetivo, a lógica proposicional torna-se importante porque nela são apresentadas as sentenças declarativas, também conhecidas como proposições. Uma proposição admite um, e somente um valor dentro de dois valores lógicos possíveis (V=verdadeiro ou F=falso). As proposições são classificadas como simples ou compostas. Baseado nisso, disserte sobre as proposições simples e as proposições compostas.

Resposta: Proposições simples também são conhecidas como proposições atômicas, e são todas as proposições consideradas unitárias. Proposição simples é aquela que não possui nenhuma outra proposição, fazendo parte de si mesma, ou seja, é composta por uma única proposição. Exemplo: Florianópolis não é a capital do Brasil. *Proposições compostas também são conhecidas como moleculares, e são formadas pela composição de duas ou mais proposições simples, em que essas proposições simples são interligadas através de palavras chamadas conectivos. Exemplo: João é médico, e Pedro é motorista.

9 – Na lógica proposicional, existem as proposições, que são formadas por um conjunto de palavras apresentando um pensamento, sendo que o valor final sempre será "verdadeiro" ou "falso". Para formar proposições compostas, são utilizadas proposições simples interligadas através de palavras que interligam essas proposições simples. Essas palavras de interligação são chamadas de conectivos. Diante disso, explique como funcionam os conectivos "E" (conjunção), "OU" (disjunção) e "Não" (Negação).

Resposta: Conectivo “E” (conjunção): quando é utilizado o conectivo E interligando duas proposições simples, por exemplo, o valor lógico final da proposição composta somente será “verdadeiro” quando as duas proposições simples forem “verdadeiras”. Se qualquer uma das proposições simples for “falsa”, ou as duas forem “falsas”, o valor lógico final da proposição composta será “falso”. *Conectivo “OU” (disjunção): quando é utilizado o conectivo OU interligando duas proposições simples, por exemplo, o valor lógico final da proposição composta será “verdadeiro” quando qualquer uma das proposições simples for “verdadeiro”, ou as duas forem “verdadeiras”. Somente quando as duas proposições simples forem “falsas”, o valor lógico final da proposição composta será “falso”. *Conectivo “Não” (negação): neste caso, o valor lógico da proposição com negação é o inverso, ou seja, será “verdadeiro” quando o valor lógico da proposição original for “falso”, e será “falso” quando o valor da proposição original for “verdadeiro”.

10 – As proposições compostas assumem as mais diversas formas e tamanhos, mas sempre são formadas por proposições simples interligadas através de conectivos lógicos. Os valores lógicos assumidos pela proposição composta podem ser "verdadeiro" ou "falso", dependendo das combinações dos valores lógicos individuais das proposições simples e do conectivo lógico utilizado, podendo ser apresentados numa tabela verdade que facilita o entendimento da proposição. No entanto, existem situações especiais com relação aos resultados da tabela verdade para a proposição

composta, chamadas Tautologia e Contradição. Assim, explique os conceitos ou características da Tautologia e da Contradição.

Resposta: Tautologia: acontece quando todos os resultados da tabela verdade de determinada proposição composta são “verdadeiros”. Isso quer dizer que para todas as combinações possíveis das proposições simples, que compõem a proposição composta, o resultado será sempre “verdadeiro”, ou seja, em nenhuma situação o resultado será “falso”. Contradição: acontece quando todos os resultados da tabela verdade de determinada proposição composta são “falsos”. Isso quer dizer que, para todas as combinações possíveis das proposições simples, o resultado da proposição composta será sempre “falso”, ou seja, em nenhuma situação o resultado será “verdadeiro”.