UNIP

EDUCAÇÃO, POR SUA ESCOLHA

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 13

CQA/UNIP – Comissão de Qualificação e Avaliação da UNIP

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 13

Christiane Mazur Doi

Doutora em Engenharia Metalúrgica e de Materiais, Mestra em Ciências - Tecnologia Nuclear, Especialista em Cálculo e Matemática Aplicada, Especialista em Estatística Aplicada e Ciência de Dados, Especialista em Língua Portuguesa e Literatura, Especialista em Recursos Gramaticais para Revisão Textual, Engenheira Química e Licenciada em Matemática, com Aperfeiçoamento em Estatística e MBA em Engenharia Econômica. Professora titular da Universidade Paulista.

Larissa Rodrigues Damiani

Doutora em Engenharia Elétrica, Mestra em Engenharia Elétrica e Engenheira Elétrica (ênfase em Telemática).

Tiago Guglielmeti Correale

Doutor em Engenharia Elétrica, Mestre em Engenharia Elétrica e Engenheiro Elétrico (ênfase em Telecomunicações). Professor titular da Universidade Paulista.

Questão 1

Questão 1. [1]

Sistemas especialistas são aqueles que fazem uso intensivo do conhecimento especializado para resolver problemas de modo semelhante àquele de um especialista humano. Uma das diversas utilizações de sistemas especialistas é na elaboração de diagnósticos destinados a inferir sobre o mau funcionamento de um sistema, a partir de observações, de modo a prescrever soluções para a anomalia detectada. A medicina, atualmente, já se beneficia do uso de sistemas especialistas em conjunto com técnicas de inferência estatística.

Uma das formas de trabalhar com sistemas especialistas é por meio do uso de "regras de produção", que são instruções do tipo "se - então". As regras de produção aplicam-se à memória de trabalho, que contém dados, e se tiverem êxito, contribuirão com alguma nova informação para a memória.

A metodologia de trabalho de um médico do serviço de emergência inclui a utilização de um sistema especialista para diagnóstico de dengue, para o qual se encontra a seguinte regra de produção.

```
Se ((temperatura > 38º) e (dores musculares intensas))
então (quadro de dengue, com 70% de chance);
senão (quadro de resfriado comum com 30% de chance);
```

Levando em conta a regra de produção do sistema especialista utilizado pelo médico, é correto afirmar que

A. cada pessoa que procura atendimento apresentando ambos os sintomas, possui a probabilidade "0,7" de estar contaminada pela dengue.
B. se 50 pessoas procurarem o atendimento apresentando pelo menos um dos dois sintomas, provavelmente 15 pessoas estarão com resfriado comum.
C. se 100 pessoas procurarem o atendimento apresentando ambos os sintomas, a quantidade de pessoas com dengue será igual a 70.
D. cada pessoa que chega apresentando um dos dois sintomas estará com resfriado comum.
E. a cada 100 pessoas que procuram atendimento, no máximo 30 estarão doentes.

[1]Questão 34 – Enade 2021 (com adaptações).

1. Análise da questão

A questão exige conhecimentos a respeito do comando condicional do tipo `se-então-senão` e de operadores lógicos, em especial, o operador "e". Vamos fazer uma breve revisão desses conteúdos.

Comando condicional `se-então-senão` e operadores lógicos

Determinado comando condicional do tipo `se` (equivalente ao `if` das linguagens de programação) testa uma proposição lógica, que chamamos de **condição**. Essa proposição pode conter operações aritméticas, relacionais ou lógicas. Caso a condição apresente valor lógico verdadeiro, o bloco de comandos identificado como `então` será executado. Caso a condição apresente valor lógico falso, o bloco de comandos identificado como `senão` será executado.

A condição a ser testada pode "misturar" operações aritméticas, relacionais e lógicas, assim como pode verificar valores de variáveis. Por exemplo, considere o comando a seguir, encontrado no enunciado.

```
Se ((temperatura > 38º) e (dores musculares intensas))
```

Encontramos a condição a ser testada entre os parênteses abertos após o comando `Se`. Nesse caso, a condição será considerada verdadeira apenas se tanto a operação relacional de `(temperatura > 38º)` quanto o valor lógico de `(dores musculares intensas)` forem verdadeiros. Isso acontece porque ambos são unidos por um operador lógico, **e** (usado na operação lógica de **conjunção**), que exige que ambos os seus operandos sejam verdadeiros para que o resultado da operação lógica seja verdadeiro.

Essa atuação pode ser resumida na tabela-verdade abaixo, em que V representa valor lógico verdadeiro e F representa valor lógico falso. Perceba que, apenas se o operando A e o operando B forem verdadeiros, o resultado da operação lógica contendo o conectivo **e** será verdadeiro.

Tabela 1. Tabela-verdade da conjunção entre o operando A e o operando B.

A	**B**	**A e B**
V	V	V
V	F	F
F	V	F
F	F	F

Outro operador lógico usado com muita frequência na programação é o operador **ou** (usado na operação lógica de **disjunção**), que exige que pelo menos um de seus operandos seja verdadeiro para que o resultado da operação lógica seja verdadeiro. Essa atuação pode ser resumida na tabela-verdade abaixo. Perceba que, apenas se o operando A e o operando B forem falsos, o resultado da operação lógica contendo o conectivo **ou** será falso.

Tabela 2. Tabela-verdade da disjunção entre o operando A e o operando B.

A	**B**	**A ou B**
V	V	V
V	F	V
F	V	V
F	F	F

2. Análise das alternativas

A – Alternativa correta.

JUSTIFICATIVA. Vamos avaliar o comando do enunciado, reproduzido abaixo.

```
Se ((temperatura > 38º) e (dores musculares intensas))
então (quadro de dengue, com 70% de chance);
senão (quadro de resfriado comum com 30% de chance);
```

Caso a condição composta pelo operador "e" seja verdadeira, isso significa que a temperatura do paciente é maior do que 38º e também que ele sente dores musculares intensas. Nesse caso, é o `então` que entra em ação, indicando 70% de chance de quadro de dengue. Isso significa que cada pessoa que procura atendimento apresentando ambos os sintomas tem probabilidade de 70% (ou de 0,7, já que $0{,}7 \cdot 100 = 70\%$) de estar contaminada pela dengue.

Essa probabilidade de 70% significa que, com base nas informações disponíveis e nas condições observadas, o sistema especialista acredita que há alta probabilidade de que a pessoa esteja com dengue. No entanto, isso não é uma garantia absoluta, pois ainda há a chance de 30% de que a pessoa não tenha dengue.

Se as condições "(temperatura > 38º) e (dores musculares intensas)" não forem atendidas, o sistema especialista executará a parte `senão` da regra. Isso significa que, se a temperatura não for maior do que 38º ou se não houver dores musculares intensas, o sistema não considerará um quadro de dengue com 70% de chance. Em vez disso, ele considerará a probabilidade de 30% para um quadro de resfriado comum.

B – Alternativa incorreta.

JUSTIFICATIVA. A regra de produção estabelece que, se 50 pessoas procurarem o atendimento, cada uma delas apresentando "apenas" um sintoma, há 30% de chance que elas tenham um resfriado comum. No entanto, a alternativa trata de pessoas que apresentam pelo menos um dos dois sintomas. Portanto, é possível que pessoas que apresentam ambos os sintomas também estejam presentes.

C – Alternativa incorreta.

JUSTIFICATIVA. Se 100 pessoas apresentarem ambos os sintomas, de acordo com a regra de produção, esperamos que 70% delas tenham dengue ($0{,}7 \cdot 100\% = 70\%$). No entanto, isso se trata apenas de uma probabilidade, não de uma regra que nos levará a uma contagem exata de diagnósticos.

D – Alternativa incorreta.

JUSTIFICATIVA. A regra de produção não diz que todas as pessoas com um dos sintomas terão resfriado comum. Ela apenas atribui a probabilidade de 30% de ter resfriado comum para aquelas que não apresentam ambos os sintomas.

E – Alternativa incorreta.

JUSTIFICATIVA. A regra de produção não fornece informações suficientes para calcular o número total de pessoas doentes a cada 100 que procuram atendimento. A probabilidade de uma pessoa estar doente depende dos sintomas apresentados.

3. Indicações bibliográficas

- DAGHLIAN, J. *Lógica e álgebra de Boole*. São Paulo: Atlas, 2012.
- BISPO, C. A. F.; CASTANHEIRA, L. B. *Introdução à lógica matemática*. São Paulo: Cengage Learning, 2011.
- BACKES, A. *Linguagem C: completa e descomplicada*. Rio de Janeiro: LTC, 2023.

Questão 2

Questão 2.[2]

Uma equipe de matemáticos foi contratada para proceder com a análise e especificação de critérios para a identificação de potenciais clientes para uma empresa de seguros. Após a análise, a equipe determinou que se deve considerar as seguintes variáveis:

a) ser maior de idade.

b) possuir residência própria ou não.

c) possuir algum parente que já possui seguro da companhia.

Em função da análise realizada, a companhia pretende estipular a viabilidade ou não do seguro, além de decidir seu preço. A equipe contratada modelou cada variável identificada utilizando as seguintes funções booleanas.

- $I(x)$: função que verifica se a pessoa x é maior de idade.
- $R(x)$: função que verifica se a pessoa x possui residência própria.
- $P(x, y)$: função que verifica se as pessoas x e y são parentes.
- $S(x)$: função que verifica se a pessoa x já possui seguro da companhia.

Após essa modelagem matemática, a seguinte tabela foi obtida. Ela expressa, utilizando a notação de lógica quantitativa, os critérios para estabelecimento de viabilidade e eventuais preços de seguros a serem concedidos.

Critério	**Resultado**
$\neg I(x)$	Inviável
$I(x) \wedge R(x) \wedge \exists y(S(y) \wedge P(x, y))$	Viável, preço: R$200,00
$I(x) \wedge \neg R(x) \wedge \exists y(S(y) \wedge P(x, y))$	Viável, preço: R$300,00
$\neg I(x) \vee (R(x) \wedge I(x))$	Viável, preço: R$500,00

A tabela foi repassada à equipe de programadores, cenário comum em que profissionais de diferentes áreas do conhecimento devem interagir a fim de obter as soluções desejadas. A solução foi implementada utilizando a linguagem C. Considere que as funções I e R foram escritas e são booleanas, operando de acordo com a definição dada pela equipe. A função "obtem" encapsula o funcionamento das funções P e S. Ela percorre a base de dados da companhia e obtém um parente da pessoa apontada por "p" que possua o seguro, caso exista. Caso contrário, ela devolve NULL.

[2]Questão 35 – Enade 2021.

```
void obtem_resultado (pessoa * p){
     pessoa*y = NULL;
     if(!I(p))
          printf(“Inviável”);
     else if(I(p)&&R(p)&&y=obtem(p))
          printf(“Viável,preço:R$200”);
     else if(I(p)&&!R(p)&&y=obtem(p))
          printf(“Viável,preço:R$300”);
     else if(!I(p)||(R(p)&&I(p)))
          printf(“Viável,preço:“R$500”);
}
```

Considerando as informações apresentadas, avalie as afirmativas.

I. A tabela apresentada é ambígua, pois há pessoas para as quais a análise resultaria em dois resultados diferentes.

II. Embora a tabela seja ambígua, não existe a possibilidade de o programa exibir mais de um resultado para uma pessoa.

III. Segundo a tabela, não existe a possibilidade de pessoas sem residência própria serem contempladas com o seguro.

IV. Segundo o funcionamento do programa, o seguro para uma pessoa menor de idade e sem residência própria é viável e lhe custará R$500.

É correto apenas o que se afirma em

A. I e II.

B. II e III.

C. III e IV.

D. I, II e IV.

E. I, III e IV.

1. Análise da questão

A questão testa conhecimentos de lógica proposicional, de lógica quantitativa e de comandos condicionais aninhados, implementados por meio da linguagem C. O que é requerido, principalmente, é que saibamos interpretar o contexto do enunciado e as expressões lógicas da tabela. Para isso, devemos conhecer a simbologia envolvida. Além disso,

precisamos ter noção da estrutura gerada por comandos do tipo `else if`, que serão comentados a seguir.

1.1. Comandos `else if`

Quando deparamos com um código, escrito em linguagem C, que apresenta uma linha do tipo `else if`, não estamos diante de um comando único, mas sim de dois comandos seguidos. Temos, simplesmente, a colocação de um comando `else` convencional, seguido de um comando `if`. Como resultado, teremos um `if` aninhado no bloco de comandos do `else` que o precede. Observe o trecho a seguir.

```
1.  if(A)
2.      X;
3.  else if(B)
4.      Y;
5.  else if(C)
6.      Z;
```

Nesse caso, `A`, `B` e `C` são condições a serem testadas, enquanto `X`, `Y` e `Z` são os seus respectivos comandos, que serão executados em caso de condição verdadeira. Vamos, a seguir, fazer a análise do trecho.

Na linha 1, é testado se a condição `A` é verdadeira. Caso seja, será executado o comando `X`, disposto na linha 2. O `else` encontrado na linha 3 está atrelado ao `if` da linha 1, ou seja, será executado apenas se a condição `A` tiver retornado valor lógico **falso**. Portanto, caso `A` seja uma condição falsa, será feito um novo teste condicional, agora para testar se a condição `B` é verdadeira. Caso seja, é executado o comando `Y`. Senão, será feito o teste da condição `C`. Se `C` for uma condição verdadeira, o comando `Z` é executado. Senão, a estrutura de aninhamento de comandos `if-else` é encerrada.

Repare que a única condição que será necessariamente testada em uma execução desse tipo é a condição `A`. A condição `B` apenas será testada caso a condição `A` tenha gerado valor lógico falso. A condição `C` apenas será testada caso tanto `A` quanto `B` tenham gerado valor lógico falso.

Isso faz com que a execução dos comandos `X`, `Y` e `Z` seja feita isoladamente. Ou seja: se `X` foi executado, `Y` e `Z` não serão; se `Y` foi executado, `X` e `Z` não serão; se `Z` foi executado, `X` e `Y` não serão. Existe, também, a possibilidade de nenhum deles ser executado, caso as três condições sejam falsas.

O fluxograma a seguir, disposto na figura 1, mostra a estrutura de aninhamento gerada pelo código do exemplo. A primeira condição, `A`, é testada em um comando `if`. A partir daí, as próximas condições serão testadas dentro do bloco `else` do comando `if` anterior. Nesse formato, caso alguma condição seja verdadeira, os testes das próximas condições serão interrompidos, já que comandos `else` são executados apenas com condições falsas. Repare que a execução de qualquer um dos comandos `X`, `Y` ou `Z` impede que os outros comandos sejam executados.

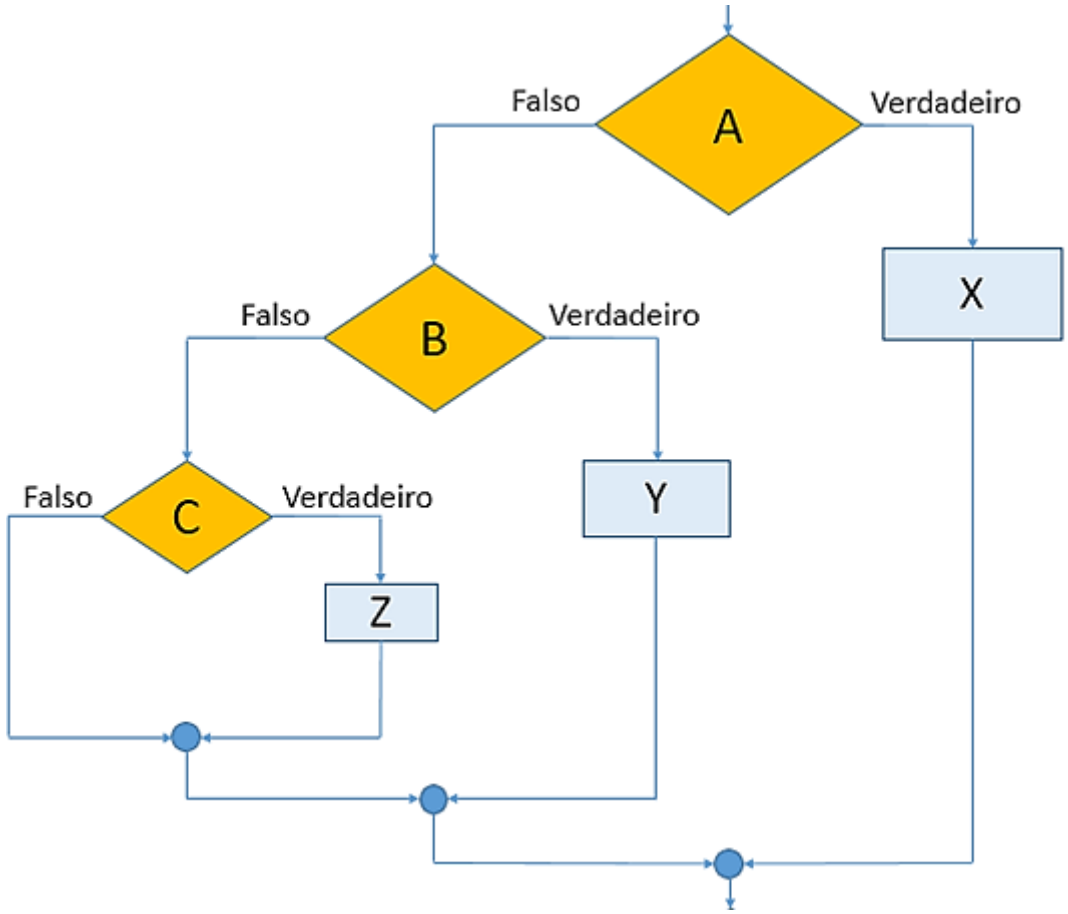


Figura 1. Fluxograma mostrando o aninhamento de comandos `if-else`, gerado pela utilização de linhas do tipo `else if`.

1.2. Operadores lógicos fundamentais

No enunciado da questão, encontramos diversos operadores lógicos, compondo expressões que identificam os critérios para o estabelecimento de viabilidade dos potenciais clientes e dos eventuais preços de seguros a serem concedidos.

Vamos, agora, relembrar o significado desses operadores, começando pelos operadores lógicos fundamentais, utilizados na lógica proposicional. Os três operadores fundamentais são:

não, **e** e **ou**. Eles são utilizados, respectivamente, nas operações lógicas de **negação**, de **conjunção** e de **disjunção**.

O operador **não**, cuja simbologia costuma ser denotada como $\sim$ ou $\neg$, inverte o valor lógico da proposição sobre a qual atua. Desse modo, se p é uma proposição verdadeira, $\neg p$ (lemos "não p") é uma proposição falsa.

O operador **e**, denotado simbolicamente por $\wedge$, atua sobre duas proposições, que são seus operandos. O resultado de sua operação de conjunção é verdadeiro apenas se ambos os seus operandos são verdadeiros. Considere que p é uma proposição verdadeira e que q também é uma proposição verdadeira. Nesse caso, a operação $p \wedge q$ (lemos "p e q") tem valor lógico verdadeiro. Se pelo menos uma de suas proposições componentes for falsa, o resultado da operação será falso.

O operador **ou**, denotado simbolicamente por $\vee$, também atua sobre duas proposições, que são seus operandos. O resultado de sua operação de disjunção é falso apenas se ambos os seus operandos são falsos. Desse modo, se pelo menos um deles é verdadeiro, o resultado da operação é verdadeiro. Considere que p é uma proposição verdadeira e que q é uma proposição falsa. Nesse caso, a operação $p \vee q$ (lemos "p ou q") tem valor lógico verdadeiro, já que uma das componentes, p, é verdadeira.

A tabela-verdade das operações lógicas discutidas, que resume a atuação dos operadores fundamentais, é apresentada a seguir, na tabela 1.

Tabela 1. Tabela-verdade das operações lógicas fundamentais.

		Negação	**Conjunção**	**Disjunção**
p	q	$\neg p$	$p \wedge q$	$p \vee q$
V	V	F	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	F

É importante destacar que, na lógica proposicional, existem outros operadores lógicos secundários, derivados dos operadores fundamentais que acabamos de relembrar, que não serão abordados neste conteúdo.

1.3. Operadores lógicos quantitativos

A lógica proposicional representa a estrutura de sentenças que utilizam os operadores fundamentais ou secundários, que não conseguem "quebrar" sentenças em partes menores nem representar quantidades associadas a elas. Para essa atuação, entramos na lógica

quantitativa, ou lógica de predicados. Nesse contexto, utilizamos os **quantificadores**, que são operadores capazes de descrever sentenças que estipulam quantidades associadas a determinada variável.

Considere que $P(x)$ é uma sentença em função da variável x. O **quantificador universal** $\forall$ (lemos "todo" ou "qualquer") expressa o fato de que, para todo elemento x, $P(x)$ será uma sentença verdadeira. De outro modo, ao dizermos $\forall x(P(x))$, estamos afirmando que, para qualquer valor assumido por x, a sentença $P(x)$ é verdadeira. Por exemplo: considere que x represente seres humanos e que a sentença $P(x)$ afirme que o ser x é mortal. Ao dizermos $\forall x(P(x))$, estamos afirmando que qualquer ser humano é mortal.

O **quantificador existencial** $\exists$ (lemos "existe" ou "há algum" ou "pelo menos um") expressa o fato de que, para pelo menos um elemento x, $P(x)$ será uma sentença verdadeira. De outro modo, ao dizermos $\exists x(P(x))$, estamos afirmando que, para algum valor assumido por x, a sentença $P(x)$ é verdadeira. Por exemplo: considere que x represente seres humanos e que a sentença $P(x)$ afirme que o ser x fala português. Ao dizermos $\exists x(P(x))$, estamos afirmando que existe pelo menos um ser humano que fala português.

As sentenças quantificadas podem ser compostas tanto por quantificadores quanto por operadores da lógica proposicional.

2. Resolução da questão

O enunciado diz que a equipe contratada modelou cada variável a ser considerada utilizando as funções booleanas a seguir.

- $I(x)$: função que verifica se a pessoa x é maior de idade.
- $R(x)$: função que verifica se a pessoa x possui residência própria.
- $P(x, y)$: função que verifica se as pessoas x e y são parentes.
- $S(x)$: função que verifica se a pessoa x já possui seguro da companhia.

A função ser "booleana" significa que ela retorna resultado ou verdadeiro ou falso. É importante notar também que, pelo contexto, podemos inferir que x representa potenciais clientes da seguradora, enquanto y representa clientes ativos da companhia.

Após essa modelagem matemática, a tabela do enunciado foi obtida. Ela expressa os critérios para o estabelecimento da viabilidade de os potenciais clientes x tornarem-se clientes efetivos, com eventuais preços de seguros a serem concedidos. A tabela é novamente reproduzida a seguir.

Tabela 2. Reprodução da tabela de critérios e resultados para potenciais clientes da seguradora, encontrada no enunciado da questão.

Critério	**Resultado**
$\neg I(x)$	Inviável
$I(x) \wedge R(x) \wedge \exists y(S(y) \wedge P(x,y))$	Viável, preço: R$200,00
$I(x) \wedge \neg R(x) \wedge \exists y(S(y) \wedge P(x,y))$	Viável, preço: R$300,00
$\neg I(x) \vee (R(x) \wedge I(x))$	Viável, preço: R$500,00

Vamos, agora, fazer a leitura de cada um dos critérios, utilizando os conhecimentos dos operadores da lógica quantitativa.

- $\neg I(x)$: a pessoa x não é maior de idade. Nesse caso, o potencial cliente é considerado inviável.
- $I(x) \wedge R(x) \wedge \exists y(S(y) \wedge P(x,y))$: a pessoa x é maior de idade, possui residência própria e existe um cliente y que é parente da pessoa x. Nesse caso, o potencial cliente é viável, com preço de R$200,00.
- $I(x) \wedge \neg R(x) \wedge \exists y(S(y) \wedge P(x,y))$: a pessoa x é maior de idade, não possui residência própria e existe um cliente y que é parente da pessoa x. Nesse caso, o potencial cliente é viável, com preço de R$300,00.
- $\neg I(x) \vee (R(x) \wedge I(x))$: pessoa x não é maior de idade, ou tem residência própria e é maior de idade. Nesse caso, o potencial cliente é viável, com preço de R$500,00.

Note que, de acordo com o primeiro critério, pessoas menores de idade devem ser classificadas como clientes inviáveis. Desse modo, o último critério é inadequado, pois testa novamente se a pessoa x é menor de idade, o que faz com que potenciais clientes menores de idade se enquadrem tanto no primeiro quanto no último critério.

No entanto, no programa desenvolvido em linguagem C, pessoas menores de idade seriam classificadas apenas como inviáveis, já que os testes de condição são feitos pelos comandos `else if`.

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. A tabela apresentada pela equipe de matemáticos é ambígua, pois pessoas menores de idade seriam consideradas tanto clientes inviáveis quanto clientes viáveis, com seguro custando R$500,00. Isso acontece porque o critério $\neg I(x)$, que significa que a pessoa

é menor de idade, aparece tanto isoladamente na primeira linha da tabela, indicando inviabilidade, quanto como um dos termos da operação de disjunção da última linha da tabela. Para a disjunção, basta que um dos seus operandos seja verdadeiro para que o resultado da operação também o seja. Desse modo, ter $\neg I(x)$ verdadeiro fará com que tanto o primeiro quanto o último critério da tabela sejam verdadeiros para a mesma pessoa x, caso ela seja menor de idade.

II – Afirmativa correta.

JUSTIFICATIVA. A codificação em linguagem C que testa as condições da tabela foi implementada por meio de comandos `else e if`, o que representa um aninhamento de comandos condicionais. A primeira condição é testada em um comando `if`. A partir daí, as próximas condições serão testadas dentro do bloco `else` do comando `if` anterior. Nesse formato, caso alguma condição seja verdadeira, os testes das próximas condições serão interrompidos, já que os comandos `else` são executados apenas com condições falsas. Desse modo, caso a condição do primeiro `if` do código seja verdadeira, o potencial cliente será classificado como inviável, e os próximos testes condicionais nem mesmo ocorrerão, por fazerem parte do bloco de comandos do `else`. Por conta disso, não existe a possibilidade de o programa exibir mais de um resultado para a mesma pessoa, mesmo que ela seja menor de idade.

III – Afirmativa incorreta.

JUSTIFICATIVA. Pessoas sem residência própria, ou seja, pessoas para as quais a condição $\neg R(x)$ é verdadeira, podem ser contempladas pela terceira condição da tabela, que indicará preço de R$300,00. A contemplação só ocorrerá se a pessoa for maior de idade e tiver algum parente que já possui o seguro da companhia, de acordo com a expressão lógica.

IV – Afirmativa incorreta.

JUSTIFICATIVA. Conforme já discutido, caso a pessoa seja menor de idade, a condição do primeiro teste `if` será verdadeira, o potencial cliente será classificado como inviável e os próximos testes condicionais nem mesmo ocorrerão.

Alternativa correta: A.

3. Indicações bibliográficas

- DAGHLIAN, J. *Lógica e álgebra de Boole*. São Paulo: Atlas, 2012.
- BISPO, C. A. F.; CASTANHEIRA, L. B. *Introdução à lógica matemática*. São Paulo: Cengage Learning, 2011.
- BACKES, A. *Linguagem C: completa e descomplicada*. Rio de Janeiro: LTC, 2023.

Questão 3

Questão 3.[3]

O sistema de numeração, inserido na arquitetura e no funcionamento dos computadores, sempre foi muito utilizado na área computacional e pode ser escrito em diferentes bases numéricas, como: binária, octal, decimal ou hexadecimal. Durante a execução dos programas, a Unidade Central de Processamento (CPU) trabalha com os dados e instruções convertidos para dois estados distintos: 0 (zero) e 1 (um), que podem ser entendidos como "com energia" e "sem energia", a chamada linguagem binária ou 0 e 1. Durante o processamento dos dados, as instruções e os dados são armazenados no formato binário na memória principal do computador. A CPU, trabalhando com dados no formato binário, aumenta sua capacidade e velocidade no processamento dos dados.

Alguns exemplos dos dados, representados em diferentes bases, são utilizados nos sistemas computacionais diariamente: o endereçamento IP dos computadores em uma rede são configurados na base decimal pontuada se for o IPv4, exemplo: 192.168.70.10; o número do endereço MAC-Address da placa de rede do computador é hexadecimal, exemplo: 00-15-5D-01-F2-00; já o sistema octal foi muito utilizado na computação, como uma alternativa mais compacta do sistema binário, na programação em linguagem de máquina.

Os profissionais que atuam na área da Tecnologia da Informação (TI) precisam constantemente interpretar, fazer cálculos ou converter dados entre as bases binária, decimal, octal ou hexadecimal. A tabela abaixo mostra os dígitos, a notação e alguns exemplos nas bases numéricas: 2, 8, 10 e 16.

Sistema na Base	Dígitos utilizados	Notação e exemplos
Binário – base 2	0, 1	$(1001)_2$; $(11011)_2$
Octal – base 8	0, 1, 2, 3, 4, 5, 6, 7	$(27)_8$; $(35)_8$; $(16)_8$
Decimal – base 10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	$(45)_{10}$; $(120)_{10}$
Hexadecimal – base 16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	$(1A)_{16}$; $(48)_{16}$; $(1E2)_{16}$

STALLINGS, W. *Arquitetura e Organização de Computadores*. São Paulo: Pearson Education do Brasil, 2017 (com adaptações).

Fazendo a conversão da soma de: $(1001)_2 + (15)_8 + (FF)_{16}$, qual é o número equivalente na base decimal?

A. $(188)_{10}$

B. $(215)_{10}$

C. $(277)_{10}$

D. $(316)_{10}$

E. $(345)_{10}$

[3]Questão 27 – Enade 2021.

1. Análise da questão

A questão testa conhecimentos a respeito da conversão de bases de numeração, já que um mesmo valor numérico pode ser escrito em diferentes bases. Vamos relembrar os quatro sistemas de numeração abordados na questão.

Bases de numeração

No nosso cotidiano, utilizamos a base 10, relativa ao **sistema decimal**. Isso significa que, no nosso sistema rotineiro, temos dez símbolos disponíveis para a representação de grandezas, que vão do 0 ao 9.

Já os sistemas computacionais, devido à constituição física do seu hardware, utilizam o **sistema binário** (de base 2) para realizar o processamento de dados e o armazenamento de informações. Isso restringe a quantidade de símbolos a apenas dois: 0 e 1.

O **sistema octal** (base 8) e o **sistema hexadecimal** (base 16) são utilizados em diversas aplicações computacionais como formas concisas de representação de dígitos binários (*bits*), já que suas bases representam potências de 2. O sistema octal é restrito a oito símbolos, que vão de 0 a 7, enquanto o sistema hexadecimal usa dezesseis símbolos, que vão de 0 a F.

Como, em base 16, precisamos de mais símbolos do que os utilizados no nosso sistema cotidiano, se convencionou adotar símbolos de letras para suprir os símbolos dos algarismos faltantes. A correspondência entre cada uma delas e a contagem em decimal é apresentada na tabela 1. Note que $(A)_{16} = (10)_{10}$, $(B)_{16} = (11)_{10}$, e assim por diante.

Tabela 1. Representação em sistema decimal dos algarismos representados por letras, do sistema hexadecimal.

Símbolo hexadecimal	Representação em decimal
A	10
B	11
C	12
D	13
E	14
F	15

2. Resolução da questão

Antes de realizarmos o somatório, vamos converter cada um dos numerais indicados para a base 10, na qual é expresso o resultado.

Vamos começar com o numeral binário, $(1001)_2$. Para convertê-lo para sistema decimal, pegamos o valor de face de cada algarismo e multiplicamos pela base original, 2, elevada ao expoente correspondente ao posicionamento do algarismo. O posicionamento começa em 0, da direita para a esquerda, e avança uma unidade a cada algarismo. Isso é exposto no quadro 1.

Quadro 1. Notação posicional do numeral 1001, escrito em sistema binário.

Posição:	3	2	1	0
Numeral em base 2:	**1**	**0**	**0**	**1**

Os cálculos da contribuição de cada algarismo são expostos a seguir.

- **Posição 0:** $1 \cdot 2^0 = 1 \cdot 1 = 1$
- **Posição 1:** $0 \cdot 2^1 = 0 \cdot 2 = 0$
- **Posição 2:** $0 \cdot 2^2 = 0 \cdot 4 = 0$
- **Posição 3:** $1 \cdot 2^3 = 1 \cdot 8 = 8$

Com isso, calculamos a contribuição individual de cada algarismo que compõe o numeral. Para sabermos o valor, em sistema decimal, do numeral como um todo, somamos todas as contribuições, conforme mostrado na sequência.

$$(1001)_2 = (1 + 0 + 0 + 8)_{10}$$

$$(1001)_2 = (9)_{10}$$

Partiremos, agora, para o numeral representado em sistema octal, $(15)_8$. Adotamos o mesmo procedimento, mas, em vez de utilizarmos base 2 nos cálculos, usaremos a base 8.

O posicionamento dos algarismos, começando em 0 à direita, é exposto no quadro 2.

Quadro 2. Notação posicional do numeral 15, escrito em sistema octal.

Posição:	1	0
Numeral em base 8:	**1**	**5**

Os cálculos da contribuição de cada algarismo são expostos a seguir.

- **Posição 0:** $5 \cdot 8^0 = 5 \cdot 1 = 5$
- **Posição 1:** $1 \cdot 8^1 = 1 \cdot 8 = 8$

Para sabermos o valor, em sistema decimal, do numeral como um todo, somamos todas as contribuições, conforme mostrado na sequência.

$$(15)_8 = (5 + 8)_{10}$$

$$(15)_8 = (13)_{10}$$

Finalmente, vamos converter o algarismo representado em sistema hexadecimal, $(FF)_{16}$. Adotamos o mesmo procedimento, mas utilizamos a base 16, nos cálculos.

O posicionamento dos algarismos, começando em 0 à direita, é exposto no quadro 3.

Quadro 3. Notação posicional do numeral FF, escrito em sistema hexadecimal.

Posição:	1	0
Numeral em base 8:	**F**	**F**

Os cálculos da contribuição de cada algarismo são expostos a seguir, considerando que F vale 15 unidades decimais.

- **Posição 0:** $15 \cdot 16^0 = 15 \cdot 1 = 15$
- **Posição 1:** $15 \cdot 16^1 = 15 \cdot 16 = 240$

Para sabermos o valor, em sistema decimal, do numeral como um todo, somamos todas as contribuições, conforme mostrado na sequência.

$$(FF)_{16} = (15 + 240)_{10}$$

$$(FF)_{16} = (255)_{10}$$

Com a conversão de todos os valores já realizada, basta somarmos todos os resultados obtidos.

$$(1001)_2 + (15)_8 + (FF)_{16} = (9)_{10} + (13)_{10} + (255)_{10}$$

$$(1001)_2 + (15)_8 + (FF)_{16} = (277)_{10}$$

Alternativa correta: C.

3. Indicações bibliográficas

- MONTEIRO, M. A. *Introdução à organização de computadores*. Rio de Janeiro: LTC, 2010.
- SZAJNBERG, M. *Eletrônica digital: teoria, componentes e aplicações*. Rio de Janeiro: LTC, 2014.
- FLOYD, T. L. *Sistemas digitais: fundamentos e aplicações*. Porto Alegre: Bookman, 2007.

Questão 4

Questão 4.[4]

Leia o texto a seguir.

O modo clássico de encarar um sistema operacional é como um gerenciador de recursos. Desse ponto de vista, o sistema operacional é responsável pelo hardware do sistema. Nesse papel, ele recebe solicitações de acesso a recursos por parte das aplicações e concede ou nega tais acessos. Ao conceder solicitações de alocação, ele deve dispor com cuidado os recursos, de modo que os programas não interfiram uns nos outros.
Por exemplo, é uma péssima ideia permitir que os programas tenham acesso sem restrição à memória uns dos outros. Se um programa com defeito (ou malicioso) escreve no espaço de memória do outro programa, o segundo programa travará, na melhor das hipóteses, ou produzirá resultados incorretos, na pior das hipóteses. Ou ainda, se o programa ofensivo modificar a memória do sistema operacional, poderá afetar o comportamento de todo o sistema.

STUART. B. L. *Princípios de sistemas operacionais: projetos e aplicações*. São Paulo: Cengage Learning, 2011 (com adaptações).

Considerando que o texto alerta para a possibilidade de um programa interferir no outro, a atividade realizada por um sistema operacional que garante essa proteção é

A. o programa antivírus.
B. o gerenciamento de memória.
C. o gerenciamento de arquivos.
D. o gerenciamento de processos.
E. o gerenciamento de entrada e saída.

1. Análise da questão

A questão aborda conhecimentos básicos a respeito das atividades exercidas pelo sistema operacional, em especial, a atividade de gerenciamento de memória. Vamos explorar brevemente esse tema.

Gerenciamento de memória

Um **sistema operacional** (SO) é um software cuja principal função é controlar o funcionamento de um computador, gerenciando a utilização e o compartilhamento dos seus recursos de hardware, como processadores, memórias e periféricos de entrada e de saída.

Sem o sistema operacional, um usuário precisaria conhecer diversos detalhes sobre hardware, o que tornaria seu trabalho lento e com grandes possibilidades de erros. Desse

[4]Questão 28 – Enade 2021.

modo, o SO funciona como uma interface entre o usuário e o hardware, tornando sua utilização mais fácil, rápida e segura.

Entre os recursos de gerenciamento oferecidos, está o **gerenciamento de memória**, que garante, por exemplo, a memória virtual – em que o SO utiliza a memória secundária como extensão da memória principal – e o acesso seguro a posições de memória por parte dos processos.

Em um ambiente de multiprogramação, ou seja, em que existe a execução em paralelo de múltiplos programas, o sistema operacional deve proteger as áreas de memória ocupadas por cada processo, além da área onde reside o próprio SO. Caso um programa tente realizar algum acesso indevido à memória, o SO deve impedi-lo. Apesar de a gerência de memória garantir a proteção de áreas da memória, mecanismos de compartilhamento devem ser oferecidos, para que diferentes processos possam trocar dados de forma protegida.

2. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. Programas antivírus detectam, impedem e agem na remoção de programas maliciosos, como vírus, trojans, worms ou spywares. São, portanto, programas usados para proteger o sistema computacional e dar mais segurança ao usuário. Programas antivírus não são, necessariamente, um recurso nativo do sistema operacional.

B – Alternativa correta.

JUSTIFICATIVA. O gerenciador de memória é um recurso do SO que tem como uma de suas atuações impedir que um programa possa interferir no espaço em memória alocado para outro programa.

C – Alternativa incorreta.

JUSTIFICATIVA. O gerenciamento de arquivos é de responsabilidade do SO, e a parte do SO que tem essa função é denominada sistema de arquivos. O sistema de arquivos é o recurso do SO responsável por gerenciar o conteúdo da memória secundária do sistema computacional, como discos rígidos e discos de estado sólido. No sistema de arquivos, estão definidas as regras para a nomeação dos arquivos, o tipo de estrutura de armazenamento, os atributos que podem ser associados a um arquivo e os procedimentos de criação, abertura e gravação em disco.

D – Alternativa incorreta.

JUSTIFICATIVA. O gerenciador de processos é o recurso do SO responsável por gerir as tarefas em execução. Desse modo, ele desempenha um importante papel no controle dos programas que estão rodando no sistema computacional, permitindo que o sistema trabalhe em modo multitarefa. Entre as funções do gerenciador de processos, está o escalonamento, no qual o SO "decide" qual processo será executado pelo processador e por quanto tempo.

E – Alternativa incorreta.

JUSTIFICATIVA. O gerenciamento de entrada e saída (E/S), em um SO, refere-se ao conjunto de funções e de subsistemas que lidam com a comunicação entre o sistema computacional e os dispositivos de entrada (como o teclado), os dispositivos de saída (como o monitor), os dispositivos híbridos (como a impressora multifuncional) e os dispositivos de armazenamento (como o disco rígido). O gerenciamento de E/S é essencial para garantir que os dispositivos funcionem eficientemente e que os programas possam interagir com eles de forma adequada.

3. Indicações bibliográficas

- MACHADO, F. B.; MAIA, L. P. *Fundamentos de sistemas operacionais*. Rio de Janeiro: LTC, 2011.
- ALVES, W. P. *Sistemas operacionais*. São Paulo: Érica, 2014.
- TANENBAUM, A. S.; WOODHULL, A. S. *Sistemas operacionais: projeto e implementação*. Porto Alegre: Bookman, 2008.

Questão 5

Questão 5.[5]

Considere a realização de uma pesquisa com 1000 pessoas para obtenção das seguintes informações:

- o valor da maior altura;
- o valor da menor altura;
- a média das alturas;
- quantas pessoas têm altura inferior à média das alturas.

Considere, ainda, que um programador foi selecionado para desenvolver um modelo de código que soluciona o problema automatizando a coleta das alturas e a geração das informações.

Com base nas informações apresentadas, desenvolva o código adequado para resolver o problema usando pseudocódigo ou uma linguagem de programação.

1. Análise da questão

A questão testa conhecimentos básicos a respeito de lógica de programação e de estatística, por meio do desenvolvimento de um código-fonte ou de um pseudocódigo. Um dos elementos que devem ser utilizados no código é o vetor, que será explorado a seguir.

Vetor

Para coletar os dados de altura de 1000 pessoas, podemos utilizar um **vetor**, já que trataremos de uma extensa série de dados do mesmo tipo. Basicamente, um vetor é uma variável indexada (*array*), que funciona como uma série de variáveis primitivas de mesmo tipo, gravadas de forma adjacente na memória principal do computador. Chamamos cada variável primitiva de **elemento** do vetor.

Em pseudocódigo, se quisermos armazenar `n` números em um vetor de nome `v`, o espaço reservado pelo vetor na memória pode ser criado pela declaração exposta a seguir, supondo que não seja necessário identificar o tipo de dado que o vetor abrigará.

```
var v[n]
```

[5]Questão Discursiva 3 – Enade 2021.

Nesse caso, `n` é um número inteiro que indica o número de elementos no vetor. Cada elemento age como uma variável primitiva, que será capaz de abrigar um valor na memória. Identificamos os elementos por um **índice**, cujos valores são representados por um número natural, que indica a posição de cada elemento no vetor. Se os números forem armazenados nas posições de índice `1, 2, …, n`, teremos um vetor de tamanho `n`, cujos índices variam de `1` a `n`.

Como exemplo, vamos considerar um vetor de nome `notas` de 5 elementos, com índices variando de 1 a 5. A declaração do vetor, em pseudocódigo, é feita a seguir, e sua representação gráfica é mostrada na figura 1.

```
var notas[5]
```

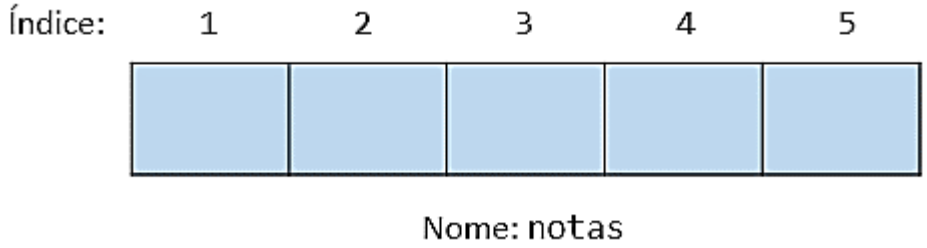


Figura 1. Representação gráfica do vetor `notas`, de 5 elementos, com valores de índice variando de 1 até 5.

Se, após a declaração, fizermos a leitura das notas de 5 alunos e armazenarmos cada uma delas em um elemento do vetor, teremos um vetor de números reais. Podemos ver, na figura 2, exemplos de notas armazenadas.

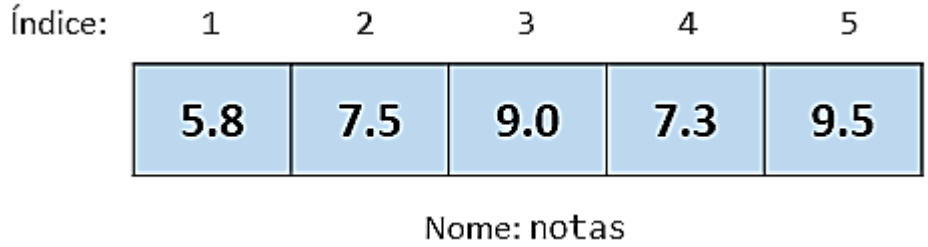


Figura 2. Representação gráfica do vetor `notas` abrigando valores distintos de notas de determinados alunos.

Para exibirmos uma dessas notas em tela, devemos identificar o nome do vetor e o índice correspondente à posição que se deseja exibir. Como exemplo, vamos considerar o comando a seguir.

```
escreva (notas[4])
```

Nesse caso, será escrito em tela o valor `7.3`, que é o valor guardado na posição de memória correspondente ao índice `4` do vetor `notas`.

É importante destacar que, em pseudocódigo, geralmente consideramos que as posições de um vetor começam em 1 e vão até o tamanho do vetor (`n`). No entanto, em linguagens de programação, é muito comum os índices iniciarem em `0` e terminarem em `n-1`. Em linguagem C, por exemplo, um vetor de 5 posições tem seus índices numerados de 0 a 4.

Para percorrer os elementos de determinado vetor, é interessante utilizarmos, em códigos-fonte, laços de repetição. O laço mais utilizado com esse intuito é o `para` (`for`).

2. Padrão de resposta do INEP

De acordo com o padrão de resposta do ENADE 2021, temos o que segue.

O respondente deve desenvolver o seguinte código:

```
algoritmo “pesquisa”
var altura[1000], menor, maior, total=0, media:real
var qtdmenor=0, ctpessoas: inteiro
inicio
leia altura[1]
menor = altura[1]
maior = altura[1]
total = altura[1]
para ctpessoas=2 até 1000 faça
  leia (altura[ctpessoas])
  total=total + altura[ctpessoas]
fimpara
media = total/1000
para ctpessoas=1 até 1000 faça
  se menor > altura[ctpessoas] então menor =
  altura[ctpessoas]
  se maior < altura[ctpessoas] então maior =
  altura[ctpessoas]
  se media > altura[ctpessoas] então qtdmenor = qtdmenor +
  1
fimpara
```

```
escreva ("Maior = ", maior, " Menor= ",menor," Média= ",media, " Quantidade de alturas menores que a média= ",qtdmenor)
```

As instruções do pseudocódigo fornecido como padrão de resposta serão brevemente comentadas nos parágrafos seguintes.

Primeiro, foram declaradas variáveis, inclusive um vetor, que serão posteriormente utilizadas para a coleta dos dados e para o armazenamento de informações. Essas variáveis são elencadas a seguir, com suas respectivas finalidades no código.

- `altura[1000]`: vetor cujos elementos são valores reais, para armazenar as alturas das 1000 pessoas.
- `menor`, `maior`: variáveis reais para armazenar a menor e a maior altura, respectivamente, encontradas ao longo do processamento do programa.
- `total`: variável real para calcular a soma de todas as alturas.
- `media`: variável real para armazenar a média aritmética simples das alturas.
- `qtdmenor`: variável inteira para "contar" quantas pessoas têm altura inferior à média aritmética das alturas.
- `ctpessoas`: variável real que contará de 2 até 1000, servindo de índice para o vetor `altura`.

Em sequência, o algoritmo inicia a leitura da altura da primeira pessoa e armazena-a na primeira posição do vetor, `altura[1]`. Depois, as variáveis `menor`, `maior` e `total` são inicializadas com os valores da primeira altura lida.

Em seguida, encontramos um laço do tipo `para`, equivalente ao `for` das linguagens de programação. O código entra em um *loop* que vai de 2 até 1000, representando os índices do vetor correspondentes à leitura das alturas das demais pessoas. Dentro do laço, ele lê cada altura e vai acumulando o somatório das alturas na variável `total`.

Após ler todas as alturas, já fora do laço `para`, o pseudocódigo calcula a média das alturas, dividindo o valor da variável `total` por 1000. Com isso, temos o somatório das alturas dividido pelo número total de pessoas, o que corresponde ao conceito de média aritmética simples da estatística básica. O resultado desse cálculo é armazenado na variável `media`.

Em seguida, encontramos um novo laço, destinado a identificar a menor e a maior altura, assim como a contar a quantidade de alturas menores do que a média. O pseudocódigo inicia esse novo laço `para`, de 1 até 1000, para percorrer todas as posições do vetor novamente. Dentro desse `loop`, ele compara a altura de cada pessoa aos valores das variáveis `menor` e

`maior` para atualizar os valores dessas variáveis, caso necessário. Além disso, ele verifica se a média aritmética é maior do que a altura da pessoa e, caso verdadeiro, incrementa em uma unidade o valor da variável `qtdmenor`.

Por último, o pseudocódigo imprime a maior altura, a menor altura, a média aritmética das alturas e a quantidade de pessoas com altura inferior à média. Desse modo, o problema proposto pela questão foi resolvido.

3. Indicações bibliográficas

- MENÉNDEZ, A. *Simplificando algoritmos*. Rio de Janeiro: LTC, 2023.
- LAMBERT, K. A. *Fundamentos de Python: estruturas de dados*. São Paulo: Cengage Learning, 2022.
- SEBESTA, R. W. *Conceitos de linguagens de programação [recurso eletrônico]*. Porto Alegre: Bookman, 2018.
- BACKES, A. *Linguagem C: completa e descomplicada*. Rio de Janeiro: LTC, 2023.

Questão 6

Questão 6.[6]

Leia o texto a seguir.

Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.

CORMEN, T. H. et al. *Algoritmos: teoria e prática*. Rio de Janeiro: Elsevier, 2002 (com adaptações).

Considere o algoritmo abaixo.

```
Algoritmo Calcular
    var
        mat [1..3][1..5] de inteiro =
        {{1, 2, -1, 2, 3}, {1, -3, 4, 2, 0}, {-3, 5, 2, 3, 4}}
        sl[1..3] de inteiro = {0, 0, 0}
        x, i, j : inteiro
        x <- 0
    início
        para i <- 1 até 3 faça
            para j <- 1 até 5 faça
                sl[i] <- sl[i] + mat[i][j]
            fimpara
            x <- x + sl[i]
        fimpara
        imprima x
    fim
```

No fim da execução do código apresentado, será exibido o valor

A. 4.

B. 7.

C. 11.

D. 22.

E. 30.

1. Análise da questão

A questão exige conhecimentos a respeito de vetores e de matrizes, utilizando um algoritmo escrito em pseudocódigo. Caso você se sinta, neste momento, desconfortável com a estrutura vetor, leia a análise da questão 5, onde há uma explicação completa a respeito desse tema. A seguir, detalharemos a estrutura matriz, que é uma variável indexada de duas dimensões.

[6]Questão 29 – Enade 2021.

1.1. Matriz

O pseudocódigo presente no enunciado traz a inicialização de uma **matriz**. Assim como um vetor, uma matriz é uma variável indexada (ou array), que funciona como uma série de variáveis primitivas de mesmo tipo, gravadas de forma adjacente na memória principal do computador. Chamamos cada variável primitiva de **elemento** da matriz.

No entanto, em um vetor, utilizamos apenas um índice de localização de elementos. Já na matriz, usamos dois índices, de forma a compor uma estrutura similar a uma tabela de dados, analogamente às matrizes matemáticas.

Em pseudocódigo, é comum considerarmos que os índices das variáveis indexadas começam em 1. Portanto, de forma resumida, uma matriz é um conjunto de variáveis de mesmo tipo, associadas ao mesmo nome, diferenciadas entre si por dois índices, cada um deles com valores iniciando em 1.

Identificamos os elementos pela combinação de dois índices, representados por números naturais, que indicam as coordenadas de cada elemento na matriz. Primeiro escolhemos o número da linha e, em seguida, escolhemos o número da coluna.

Vamos, agora, ver o exemplo de declaração de uma matriz em pseudocódigo. Faremos a declaração de uma matriz de nome `prodVendidos`, destinada a abrigar o número de produtos vendidos por um fabricante no 1º bimestre (janeiro e fevereiro) de 3 anos consecutivos (2021, 2022 e 2023).

```
var prodVendidos[1..2][1..3] de inteiro
```

Nesse caso, declaramos uma matriz de 2 linhas e 3 colunas (de ordem 2×3), sendo que cada elemento é uma variável do tipo `inteiro`, adequada para guardar um valor do conjunto matemático dos números inteiros. A estrutura dessa matriz, que abriga 6 elementos, é mostrada na figura 1.

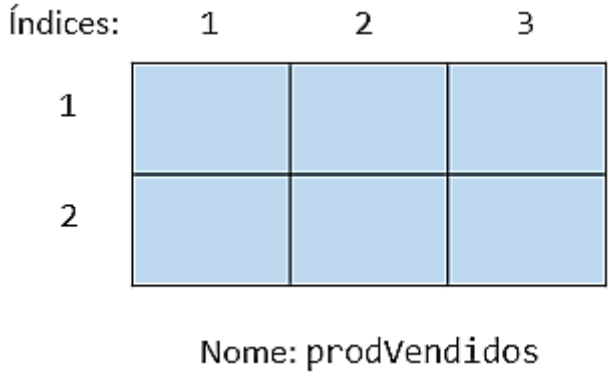


Figura 1. Representação gráfica da matriz `prodVendidos`, de 6 elementos, com valores de índice de linha variando de 1 a 2 e com valores de índice de coluna variando de 1 a 3.

Podemos, também, **inicializar** uma matriz. Isso significa que, ao declarar a matriz, podemos listar os valores que devem ser posicionados em cada elemento, no mesmo comando da própria declaração. Abaixo, temos um exemplo de inicialização da matriz `prodVendidos`, cujo contexto é o mesmo do exemplo anterior.

```
var prodVendidos[1..2][1..3] de inteiro = {{23,75,78},{82,81,94}}
```

Após a execução do comando de inicialização mostrado acima, teremos a estrutura da figura 2. Nela, vemos, também, o contexto adotado para cada linha e para cada coluna, no exemplo que estamos considerando.

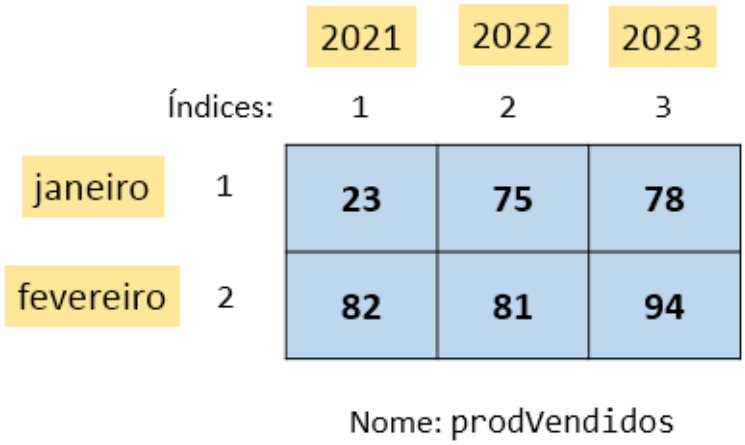


Figura 2. Representação gráfica da matriz `prodVendidos` inicializada, abrigando números distintos de produtos vendidos em determinados períodos.

Se quisermos, por exemplo, saber o número de produtos vendidos no mês de janeiro do ano de 2022, devemos buscar o elemento `prodVendidos[1][2]`, cujo valor é 75.

Para percorrer os elementos da matriz, um a um, para leitura ou para escrita de dados, é usual utilizarmos uma estrutura de dois laços `para` aninhados. O código de nome `ImprimeMatriz`, mostrado abaixo como exemplo, faz a impressão em tela, por meio do comando de saída de dados `imprima` (análogo ao comando `escreva`), dos valores da matriz `prodVendidos`, inicializada na linha 3.

```
1.  Algoritmo ImprimeMatriz
2.  var
3.      prodVendidos[1..2][1..3] de inteiro = {{23,75,78},{82,81,94}}
4.      i, j: inteiro      //i é índice de linha e j é índice de coluna
5.  início
6.      para i <- 1 até 2 faça
7.          para j <- 1 até 3 faça
8.              imprima prodVendidos[i][j]
9.          fimpara
10.     fimpara
11. fim
```

Como a matriz tem duas dimensões, precisamos usar duas variáveis (`i` e `j`) para indexar cada uma delas. A variável `i` representa o índice de linha, e a variável `j` representa o índice de coluna. Usamos um comando `para` aninhado em outro. O laço externo (da linha 6) "entra" em uma linha da matriz, e o laço interno (da linha 7) percorre todas as colunas para aquela linha. Como resultado, esse código percorre os elementos da matriz na seguinte ordem: `[0][0]`, `[0][1]`, `[0][2]`, `[1][0]`, `[1][1]`, `[1][2]`. A cada novo elemento no qual o programa "chega", é feita a exibição em tela do valor desse elemento, por meio do comando `imprima`.

1.2. Análise do código-fonte do enunciado

O pseudocódigo do enunciado será reproduzido a seguir, com as linhas numeradas, para facilitar a sua análise.

```
1.  Algoritmo Calcular
2.      var
3.          mat [1..3][1..5] de inteiro =
4.          {{1, 2, -1, 2, 3}, {1, -3, 4, 2, 0}, {-3, 5, 2, 3, 4}}
5.          sl[1..3] de inteiro = {0, 0, 0}
6.          x, i, j : inteiro
7.          x <- 0
8.      início
9.          para i <- 1 até 3 faça
10.             para j <- 1 até 5 faça
11.                 sl[i] <- sl[i] + mat[i][j]
12.             fimpara
13.             x <- x + sl[i]
14.         fimpara
15.         imprima x
16.     fim
```

O algoritmo `Calcular` tem uma seção de declaração de variáveis globais, que começa na linha 2 e se estende até a linha 7. Nessa seção, temos as declarações elencadas a seguir.

- Inicialização de uma matriz de inteiros de ordem 3×5, de nome `mat`, cujos valores dos elementos foram listados na linha 4.
- Inicialização de um vetor de inteiros de 3 posições, de nome `sl`, cujos 3 elementos assumem valor 0.
- Inicialização da variável inteira `x` com o valor 0.
- Declaração das variáveis inteiras `i` e `j`, que serão usadas posteriormente como índices de linha e de coluna dos arrays.

Na linha 8, é aberta a função principal do código, que se encerra na linha 16. Dentro dessa função, encontramos um encadeamento de laços `para`, que têm a intenção de percorrer os elementos do vetor `sl` e da matriz `mat`. O índice `i`, que percorre as linhas tanto da matriz quanto do vetor, assume do valor 1 até o valor 3. O índice `j`, que percorre as colunas da matriz, assume do valor 1 até o valor 5.

A cada execução do laço interno, o comando `sl[i] <- sl[i] + mat[i][j]` é executado. Esse comando atribui ao elemento do vetor `sl` na posição `[i]` a soma do seu próprio valor anterior com o valor do elemento da matriz na posição `[i][j]`. Isso faz com que o vetor abrigue o somatório dos elementos de cada linha da matriz.

Ao terminar as iterações laço interno, é executado o comando `x <- x + sl[i]`, que faz parte do bloco de comandos do laço externo. Nesse comando, é atribuído à variável `x` o seu próprio valor anterior somado ao elemento `[i]` do vetor. Isso faz com que a variável `x` abrigue o somatório dos elementos do vetor.

Para cada umas das 3 execuções do laço `para` externo, são feitas 5 execuções do laço `para` interno. Isso faz com que cada execução externa entre em uma linha da matriz e as execuções internas percorram as colunas daquela linha. A sequência de execuções dos laços `para` aninhados é mostrada no quadro 1. Cada execução dessas instruções dentro do laço é chamada de **iteração**.

Quadro 1. Sequência de execuções aninhadas dos laços `para` externo e interno do pseudocódigo.

Laço externo `para i <- 1 até 3 faça`	**Laço interno** `para j <- 1 até 5 faça`
Iteração 1	Iteração 1
	Iteração 2
	Iteração 3
	Iteração 4
	Iteração 5
Iteração 2	Iteração 1
	Iteração 2
	Iteração 3
	Iteração 4
	Iteração 5
Iteração 3	Iteração 1
	Iteração 2
	Iteração 3
	Iteração 4
	Iteração 5

Perceba que cada iteração do laço externo executa 5 iterações do laço interno. Com isso, no total, serão feitas 15 iterações do laço interno.

Por fim, na linha 15, é impresso em tela o valor final da variável `x`. Como dito, esse valor corresponde ao somatório dos elementos do vetor, que, por sua vez, correspondem ao somatório dos elementos de cada linha da matriz. Logo, em sua situação final, a variável `x` abrigará o somatório de todos os elementos da matriz `mat`.

2. Resolução da questão

Para resolver a questão, devemos prever a saída gerada após a execução do algoritmo. Vamos, primeiro, considerar o cenário inicial das variáveis `mat`, `sl` e `x`, gerado logo após a inicialização de cada uma delas. Essa situação é apresentada na figura 3.

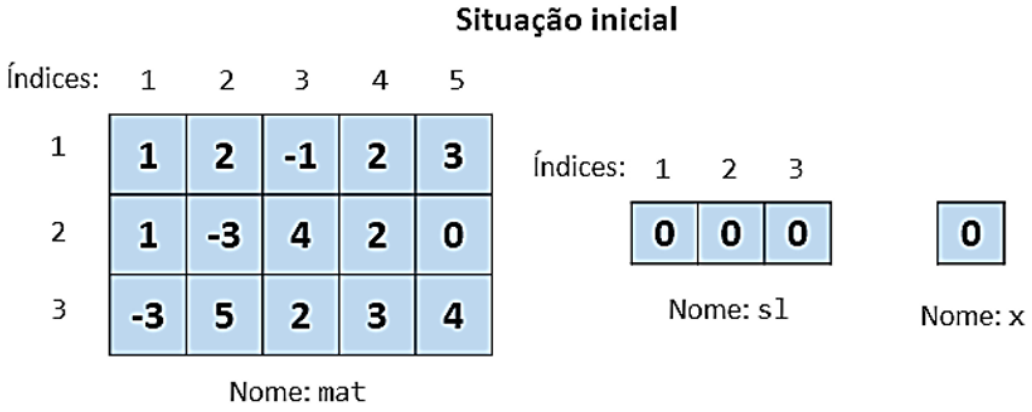


Figura 3. Situação inicial (logo após a inicialização) da matriz `mat`, do vetor `sl` e da variável `x`.

As execuções dos laços `para` farão com que o somatório dos elementos da linha 1 de `mat` seja recebido pelo elemento 1 do vetor `sl`. O mesmo ocorre para as linhas 2 e 3, cujos somatórios serão recebidos, respectivamente, pelos elementos 2 e 3 de `sl`. Desse modo, temos a situação a seguir.

`sl[1]` = 1+2+(-1)+2+3 = 7

`sl[2]` = 1+(-3)+4+2+0 = 4

`sl[3]` = -3+5+2+3+4 = 11

Ao final de cada sequência de iterações do laço `para` interno, a variável `x` vai acumulando o somatório dos elementos de `sl`. Assim, para a variável `x`, temos o cenário final a seguir.

`x` = 7+4+11 = 22

A situação final da matriz, do vetor e da variável `x` é exposta na figura 4.

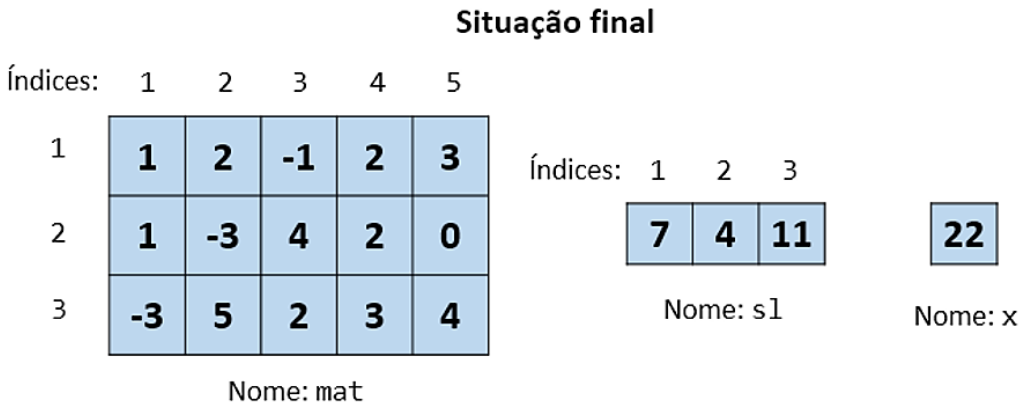


Figura 4. Situação final da matriz `mat`, do vetor `sl` e da variável `x`.

Logo, o valor exibido pelo comando da linha 15 do pseudocódigo é 22.

Alternativa correta: D.

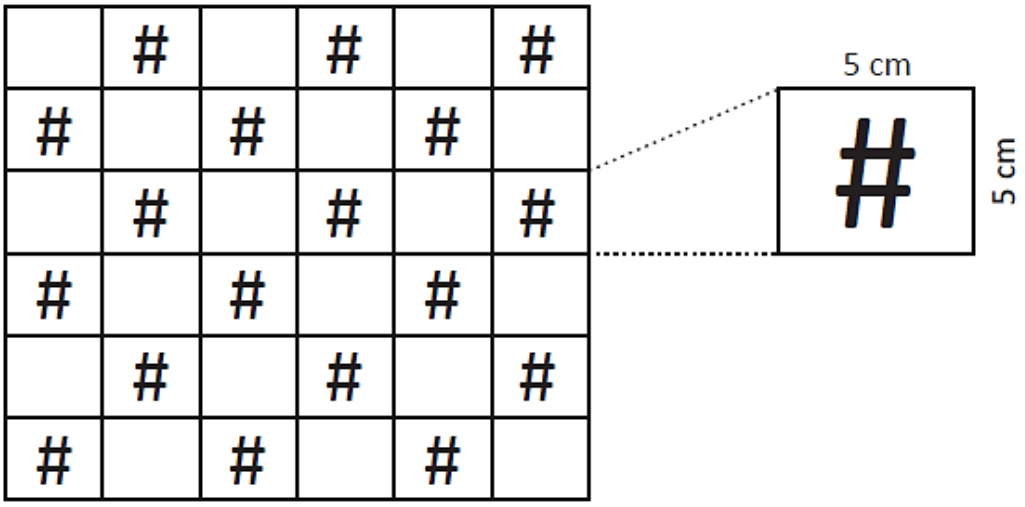
3. Indicações bibliográficas

- LAMBERT, K. A. *Fundamentos de Python: estruturas de dados*. São Paulo: Cengage Learning, 2022.
- MENÉNDEZ, A. *Simplificando algoritmos*. Rio de Janeiro: LTC, 2023.
- SEBESTA, R. W. *Conceitos de linguagens de programação [recurso eletrônico]*. Porto Alegre: Bookman, 2018.
- BACKES, A. *Linguagem C: completa e descomplicada*. Rio de Janeiro: LTC, 2023.

Questão 7

Questão 7.[7]

Uma fábrica adquiriu máquinas de costura industrial para a produção de jogos de toalhas de mesa e guardanapos de tecido com estampas quadriculadas. Cada quadrículo tem tamanho padrão de 5 cm X 5 cm. Para a configuração da costura, é preciso um programa de computador para criar um molde em que sejam informadas as quantidades de quadrículos por comprimento e por largura, além do desenho da estampa a ser costurada. Por exemplo, para um guardanapo de tecido que tenha 30 cm X 30 cm, com uma estampa representada pelo caractere "#", o molde a seguir precisa ser criado.



Como uma toalha pode ter, no máximo, 250 cm de comprimento e 200 cm de largura, o programador declarou uma matriz `char molde[40][50]` para ser possível representar moldes cujas medidas compreendam esses limites. Ele também implementou, entre outras, as seguintes funções:

- `criar_molde`, para a qual os seguintes parâmetros são esperados, nessa ordem: uma referência à matriz molde, o comprimento e a largura do tecido (em quantidades de quadrículos) e o caractere que representa a estampa;
- `alternar_estampa`, que tem a tarefa de controlar a disposição alternada das estampas no molde.

Considerando o cenário descrito, que alternativa a seguir apresenta a implementação correta das funções `criar_molde` e `alternar_estampa`?

[7]Questão 10 – Enade 2021.

A.
```
char alternar_estampa(char estampa, char *proxima) {
  *proxima = *proxima == estampa ? ' ' : estampa;
  return *proxima;
}
void criar_molde(char molde[][50], int c, int l, char estampa) {
  char proxima = estampa;
  for (int i = 0; i < l; i++)
      for (int j = 0; j < c; j++)
          molde[i][j] = alternar_estampa(estampa, &proxima);
}
```

B.
```
char alternar_estampa(char estampa, char proxima) {
  proxima = proxima == estampa ? ' ' : estampa;
  return proxima;
}
void criar_molde(char molde[][50], int c, int l, char estampa) {
  char proxima = estampa;
  for (int i = 0; i < l; i++)
      for (int j = 0; j < c; j++)
          molde[i][j] = alternar_estampa(estampa, proxima);
}
```

C.
```
char alternar_estampa(char estampa, char *proxima) {
  *proxima = *proxima == estampa ? ' ' : estampa;
}
void criar_molde(char molde[][50], int c, int l, char estampa) {
  char proxima = estampa;
  for (int i = 0; i < l; i++)
      for (int j = 0; j < c; j++)
          molde[i][j] = alternar_estampa(estampa, &proxima);
}
```

D.
```
char alternar_estampa(char estampa, char *proxima) {
  *proxima = *proxima == estampa ? ' ' : estampa;
  return *proxima;
}
void criar_molde(char molde[][50], int c, int l, char estampa) {
  char proxima = estampa;
  for (int i = 0; i < l; i++) {
      for (int j = 0; j < c; j++)
          molde[i][j] = alternar_estampa(estampa, &proxima);
      próxima = molde[i][0];
  }
}
```

E.
```
char alternar_estampa(char estampa, char proxima) {
  proxima = proxima == estampa ? ' ' : estampa;
  return proxima;
}
void criar_molde(char molde[][50], int c, int l, char estampa) {
  char proxima = estampa;
  for (int i = 0; i < l; i++) {
      for (int j = 0; j < c; j++)
          molde[i][j] = alternar_estampa(estampa, proxima);
      próxima = molde[i][0];
  }
}
```

1. Análise da questão

Para resolvermos essa questão, precisamos aplicar conhecimentos a respeito de matrizes, de funções e de ponteiros, implementados por meio da linguagem C. Além disso, devemos conhecer a notação de condicional ternária, explicada a seguir.

1.1. Condicional ternária

Em linguagem C, os operadores "`?`" e "`:`" são utilizados como condicional ternária. Por meio deles, podemos escrever um comando `if-else` de forma reduzida. Observe o quadro abaixo, onde representamos a condição do comando `if` por `X`, o bloco de comandos do `if` por `Y` e o bloco de comandos do `else` por `Z`.

Quadro 1. Equivalência entre o comando `if-else` convencional e a condicional ternária.

`if(X) {`
`  Y`
`}`
`else {`
`  Z`
`}`
	⇔	`X ? Y : Z`

À esquerda, vemos a representação convencional, com os comandos `if` e `else` escritos por extenso, com seus blocos de comandos envoltos por pares de chaves. À direita, temos uma expressão condicional ternária, utilizando os operadores "?" e ":", o que equivale a todas as linhas de código vistas à esquerda. Nesse caso, a condição do `if` aparece antes de "?", o bloco de comando do `if` aparece após "?" e o bloco de comandos do `else` é posicionado após o operador ":".

Uma expressão condicional ternária pode ser aplicada em um comando de atribuição de valores a uma variável. Como exemplo, considere o código, escrito em linguagem C, exposto a seguir.

```
7.  #include <stdio.h>
8.  int main(void) {
9.       char caractere;
10.      int i=1;
11.      caractere = i==0 ? 'b' : 'a';
12.      printf("%c", caractere);
13.      return 0;
14. }
```

Na linha 5, temos a atribuição de valores atrelada a um teste condicional. Se for verdade que `i` vale 0, será atribuído o caractere `'b'` ao conteúdo da variável `caractere`. Senão, será atribuído o caractere `'a'` ao conteúdo da variável `caractere`. Nesse caso, a execução do programa levará até a impressão da letra "a", pois não é verdade que `i` vale 0.

Na questão, a condicional ternária é usada nas funções expostas nas alternativas, e sua sintaxe deve ser reconhecida para que a questão seja resolvida.

1.2. Matriz

Para organizar os quadrículos da toalha, conforme o contexto da questão exige, devemos utilizar uma **matriz**, estrutura já abordada na análise da questão 6. Nessa seção, traremos a aplicação dessa estrutura conforme a implementação na sintaxe da linguagem C.

Uma matriz é uma variável indexada (ou array), que funciona como uma série de variáveis primitivas de mesmo tipo, gravadas de forma adjacente na memória principal do computador. Chamamos cada variável primitiva de **elemento** da matriz.

Na matriz, são utilizados dois índices de identificação de elementos, de forma a compor uma estrutura similar a uma tabela de dados, analogamente às matrizes matemáticas. Nas linguagens de programação, cada um desses índices é iniciado com o valor 0.

Logo, em linguagem C, uma matriz é um conjunto de variáveis de mesmo tipo, associadas ao mesmo nome, diferenciadas entre si por dois índices, cada um deles com valores iniciando em 0. A declaração é feita pela sintaxe a seguir.

```
tipo_dado nome_matriz[nº_linhas][nº_colunas];
```

Identificamos os elementos pela combinação de dois índices, representados por números naturais, que indicam as coordenadas de cada elemento na matriz.

Como exemplo, vamos declarar uma matriz de nome `lucro` para abrigar o lucro no 1º bimestre (janeiro e fevereiro) de 3 anos consecutivos (2021, 2022 e 2023).

```
float lucro[2][3];
```

Nesse caso, declaramos uma matriz de 2 linhas e 3 colunas (de ordem 2×3), sendo que cada elemento é uma variável do tipo `float`, adequada para guardar um valor real. A estrutura dessa matriz, que abriga 6 elementos, é mostrada na figura 1.

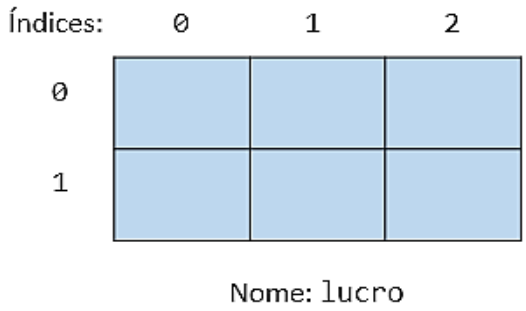


Figura 1. Representação gráfica da matriz `lucro`, de 6 elementos, com valores de índice de linha variando de 0 a 1 e com valores de índice de coluna variando de 0 a 2.

Se, após a declaração, fizermos a leitura dos lucros de 6 períodos e armazenarmos cada uma delas em um elemento da matriz, teremos a estrutura mostrada na figura 2, que ilustra exemplos de valores de lucro armazenados. Na figura, vemos, também, o contexto adotado para cada linha e para cada coluna, no exemplo que estamos considerando.

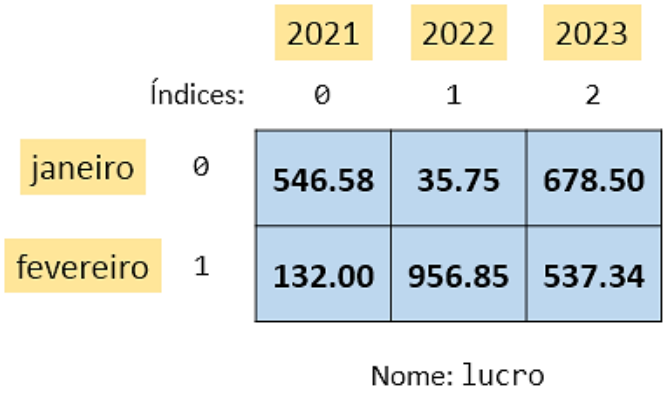


Figura 2. Representação gráfica da matriz `lucro` abrigando valores distintos de lucros de determinados períodos.

Se quisermos, por exemplo, saber o valor do lucro do mês de janeiro do ano de 2022, deveremos buscar o elemento `lucro[0][1]`, cujo valor é `35.75`.

Para percorrer os elementos da matriz, um a um, para a leitura ou para a escrita de dados, é usual utilizarmos uma estrutura de dois laços `for` aninhados. O código abaixo, por exemplo, faz a leitura pelo teclado, por meio do comando `scanf`, dos valores de uma matriz que abriga números reais.

```
1.  #include <stdio.h>
2.  int main(void){
3.      float matriz[2][3];
4.      int i, j; //i é índice de linha e j é índice de coluna
5.      for(i=0; i<2; i++){
6.          for(j=0; j<3; j++){
7.              scanf("%f", &matriz[i][j]);
8.          }
9.      }
10.     return 0;
11. }
```

Como a matriz tem duas dimensões, precisamos usar duas variáveis (`i` e `j`) para indexar cada uma delas. A variável `i` representa o índice de linha, e a variável `j` representa o índice de coluna. Usamos um `for` aninhado em outro. O laço externo (da linha 5) "entra" em uma linha da matriz, e o laço interno (da linha 6) percorre todas as colunas para aquela linha. Como resultado, esse código percorre os elementos da matriz na seguinte ordem: `[0][0]`, `[0][1]`, `[0][2]`, `[1][0]`, `[1][1]`, `[1][2]`. A cada novo elemento no qual o programa "chega", é solicitado um valor real ao usuário, por meio do comando `scanf`.

1.3. Função

A questão exige a implementação de duas **funções**. De forma simplificada, uma função é um "bloco de código" que pode ser nomeado e chamado por um programa. Podemos criar as nossas próprias funções, de forma a modularizarmos nosso código em linguagem estruturada.

Programas grandes e complexos podem ser construídos bloco a bloco, com a ajuda de funções específicas. Desse modo, nem todos os comandos precisam ser escritos na função `main()`. Ela, no entanto, será responsável por "chamar" ou "invocar" as funções específicas criadas, de forma a colocar seus blocos de comando em execução.

Em linguagem C, a declaração de uma função segue a forma geral:

```
tipo_retornado nome_função(lista_parâmetros) {
    comandos;
    return ...;
}
```

As partes componentes da forma geral são elencadas e descritas a seguir.

- **Retorno** (`return`): maneira como a função "devolve" o resultado da sua execução para quem a chamou. O retorno não é obrigatório, sendo que, no caso de não existir, a função vai apenas executar uma sequência de comandos definida no seu bloco.
- **Tipo retornado**: tipo do valor que será "devolvido" à linha que invocou a função, por meio do comando `return`. Se não há retorno, o tipo é `void`.
- **Lista de parâmetros**: todas as variáveis locais (que só existem dentro dessa função) que recebem os argumentos passados no momento da chamada da função. Quando há mais de um parâmetro, eles são separados entre si por vírgula. Quando não há parâmetros, é usual inserir o termo `void` entre os parênteses.

Para tornar mais clara a explicação, vamos acompanhar o exemplo a seguir.

```
1.  #include <stdio.h>
2.
3.  int square(int a) {
4.      return a*a;
5.  }
6.
7.  int main(void){
8.      int num, numQuadr;
9.      num = 3;
10.     numQuadr = square(num);
11.     printf("Quadrado de %d é %d", num, numQuadr);
12.     return 0;
13. }
```

Entre as linhas 3 e 5, a função `square()` é implementada, com um parâmetro inteiro de nome `a`. Note que a execução da função, no entanto, não ocorre no início da execução do programa, pois, inicialmente, a função ainda não foi invocada.

A função `main()` é implementada, entre as linhas 7 e 13, como a função principal do programa. A execução do programa começa sempre por aqui. Duas variáveis inteiras, `num` e `numQuadr`, são declaradas na função `main()`. A variável `num` recebe o valor 3.

Na linha 10, a função `square()` é invocada, passando o valor de `num` como argumento. Um argumento é um dado "transferido" para um dos parâmetros da função invocada. Da linha 10, a execução do programa "pula" para a linha 3, já que a função `square()` foi chamada.

Na função `square()`, o parâmetro `a` recebe o argumento 3 (que era o valor de `num`). Desse modo, a função `square()` inicia sua execução com uma variável local `a`, cujo valor é 3. O cálculo do quadrado `a*a` é realizado, multiplicando o valor de `a` por ele mesmo.

O valor resultante da operação `a*a`, que é 9, é retornado da função `square()` para o ponto de invocação da função. Desse modo, devemos voltar para a linha 10. O valor 9, retornado pela função `square()`, é finalmente atribuído à variável `numQuadr`.

Em seguida, a função `printf()` é chamada para imprimir a mensagem formatada na tela, usando os valores de `num` e `numQuadr`.

Por fim, a função `main()` retorna 0 ao sistema operacional, indicando que o programa foi executado com sucesso.

Em resumo, mesmo com a função `square()` sendo definida antes da função `main()`, a execução real da função `square()` ocorre apenas quando ela é chamada, na linha 10. A função `printf()`, por sua vez, é chamada após o cálculo do quadrado e a atribuição do valor retornado a `numQuadr`.

Mas por que não tivemos que definir a função `printf()` no código, se precisamos definir a função `square()`? A função `printf()` não precisa ser definida explicitamente porque ela é parte de uma biblioteca padrão do C, acessada pela interface `stdio.h`. Por isso, incluímos a diretiva `#include <stdio.h>` no início do código. Essa biblioteca já contém a definição e a implementação da função `printf()`, juntamente com outras funções de entrada e de saída. A função `square()`, por outro lado, é uma função definida pelo programador, ou seja, precisamos fornecer a sua definição e a sua implementação no próprio código para que o compilador saiba como executá-la.

1.4. Ponteiros

Na questão, ponteiros são utilizados como parâmetros das funções. Um **ponteiro** é uma variável destinada a abrigar o **endereço de memória** de outra variável, e não o conteúdo dessa outra variável. Dizemos que um ponteiro "aponta" para uma variável, quando abriga o endereço dela.

Podemos utilizar ponteiros para acessar uma variável local em diferentes partes de um programa, na manipulação de *arrays*, no uso de arquivos ou na criação de algoritmos de estruturas de dados. Ponteiros são amplamente utilizados em linguagem C, conferindo um grande "poder" de acesso ao *hardware* ao programador.

A seguir, vemos a sintaxe básica de declaração de um ponteiro, utilizando linguagem C.

```
tipo_variável *nome_ponteiro;
```

As partes componentes da forma geral são elencadas e descritas a seguir.

- **`tipo_variavel`**: é o tipo de dado da variável cujo endereço será armazenado.
- **`*nome_ponteiro`**: é o nome escolhido para o ponteiro, que deve ser precedido por um asterisco.

Como exemplo, vamos declarar um ponteiro de nome `p`, que deve apontar para (guardar o endereço de) uma variável do tipo `int`. Nesse caso, podemos inserir o comando a seguir.

```
int *p;
```

Ponteiros também podem ser declarados como parâmetros de funções em nossos códigos. Nesse caso, dizemos que estamos trabalhando com parâmetros por **referência**. Na

passagem de argumentos, não transferimos o valor da variável para o parâmetro da função, mas sim o seu endereço.

Quando passamos um *array* (como um vetor ou uma matriz) como argumento, sempre estaremos passando um argumento por referência. O nome do *array* indica o endereço da sua primeira posição de memória. O que o parâmetro recebe, portanto, é o endereço de memória da posição inicial da variável estruturada.

Para declararmos como argumento um ponteiro `v`, destinado a receber o endereço inicial de um vetor de determinado tipo, devemos incluir, na lista de parâmetros da função, um dos dois comandos de declaração a seguir, que exercem a mesma tarefa.

```
tipo *v
```

```
tipo v[]
```

Para declararmos como argumento um ponteiro `mat`, destinado a receber o endereço inicial de uma matriz de determinado tipo, devemos incluir, na lista de parâmetros da função, o comando de declaração a seguir.

```
tipo mat[][nº_col]
```

No caso de matrizes, precisamos especificar, na declaração do ponteiro, o tamanho da 2ª dimensão, que representa o número de colunas. É opcional especificar o tamanho da 1ª dimensão, que representa o número de linhas da matriz.

2. Resolução da questão

Voltando ao contexto da questão, é dito, no enunciado, que o programador declarou uma matriz `char molde[40][50]` para ser possível representar moldes de toalhas cujas medidas compreendam os limites de dimensão especificados, com, no máximo, 250 cm de comprimento e 200 cm de largura.

Levando em consideração que cada quadrículo tem 5 cm X 5 cm, podemos ter, no máximo, 50 quadrículos no comprimento e 40 quadrículos na largura de cada toalha. Logo, a matriz declarada utiliza seu primeiro índice para largura e seu segundo índice para comprimento, obedecendo à sintaxe: `molde[largura][comprimento]`.

A respeito da função `criar_molde()`, que tem a tarefa de criar o molde da toalha a ser fabricada pela máquina de costura industrial, o enunciado indica que são esperados os quatro parâmetros elencados a seguir, em respectiva ordem.

- Uma referência à matriz molde: nesse parâmetro, esperamos encontrar um ponteiro capaz de receber o endereço de memória correspondente à primeira posição da matriz molde. A declaração desse parâmetro é dada no formato `char molde[][50]`, já que a matriz molde, que será passada à função por referência, é do tipo `char` e o tamanho da segunda dimensão da matriz é obrigatório.
- O comprimento do tecido (em quantidades de quadrículos): aqui, esperamos encontrar uma variável do tipo `int`, destinada a receber o número inteiro correspondente à contagem de quadrículos do comprimento da toalha. A declaração desse parâmetro é dada no formato `int c`.
- A largura do tecido (em quantidades de quadrículos): nesse parâmetro, esperamos encontrar uma variável do tipo `int`, destinada a receber o número inteiro correspondente à contagem de quadrículos da largura da toalha. A declaração desse parâmetro é dada no formato `int l`.
- O caractere que representa a estampa: nesse parâmetro, esperamos encontrar uma variável do tipo `char`, destinada a receber o caractere que irá compor a estampa da toalha. A declaração desse parâmetro é dada no formato `char estampa`.

A criação do molde não necessita de um retorno, pois a função apenas executa uma tarefa descrita em seu bloco de comandos. Logo, a função `criar_molde()` será declarada como sendo do tipo `void`, obedecendo à sintaxe exposta a seguir.

```
void criar_molde(char molde[][50], int c, int l, char estampa)
```

A respeito da função `alternar_estampa()`, o enunciado apenas diz que ela tem a função de controlar a disposição alternada das estampas no molde. Desse modo, esperamos que a função `criar_molde()`, em seu bloco de comandos, invoque a função `alternar_estampa()`, já que o controle da disposição é uma das etapas da criação do molde. O código completo de uma possível implementação das funções `alternar_estampa()` e `criar_molde()` é disposto a seguir, com a invocação ocorrendo na linha 10.

```
1.  char alternar_estampa(char estampa, char *proxima) {
2.      *proxima = *proxima == estampa ? ' ' : estampa;
3.      return *proxima;
4.  }
5.
6.  void criar_molde(char molde[][50], int c, int l, char estampa) {
7.      char proxima = estampa;
8.      for (int i = 0; i < l; i++) {
9.          for (int j = 0; j < c; j++)
```

```
10.              molde[i][j] = alternar_estampa(estampa, &proxima);
11.          proxima = molde[i][0];
12.      }
13.  }
```

Ao ser chamada, a função `criar_molde()` recebe o endereço inicial da matriz, o número de colunas, o número de linhas e o caractere a ser utilizado como estampa. Em seguida, na linha 7, é declarada uma variável denominada `proxima` que recebe, inicialmente, o valor do caractere de `estampa`.

As linhas 8 e 9 mostram o aninhamento de dois laços `for`, que servirão para percorrer a matriz. O `for` da linha 8 percorre as linhas, enquanto o `for` da linha 9 percorre as colunas da matriz. Isso permite que o programa percorra a matriz da posição [0][0] até a última posição da matriz, preenchendo cada posição com o caractere adequado para gerar a estampa alternada.

Na linha 10, temos um comando que pertence ao bloco de comandos do laço `for` interno, destinado a percorrer todas as colunas de determinada linha da matriz. É atribuído à posição atual da matriz o retorno da função `alternar_estampa()`. A invocação dessa função é feita, passando como argumentos o caractere de `estampa` e o endereço da variável `proxima`.

Agora, a função `alternar_estampa()` entra em ação. Para controlar a disposição alternada, a função deve manter um quadrículo em branco e outro com a impressão da estampa, conforme pode ser visto na figura do enunciado. Para isso, a função deve "saber" qual é o caractere desejado para estampar a toalha. Com o intuito de receber esse caractere, a função deve ter um parâmetro, que pode ser declarado como `char estampa`.

Além disso, ela também deve saber o endereço da variável `proxima`, que indica qual deverá ser a próxima impressão na matriz. Para isso, é necessário um segundo parâmetro, que deve ser um ponteiro e pode ser declarado como `char *proxima`. Ao passar o endereço da variável `proxima` como argumento para a função `alternar_estampa()`, é possível acessar e modificar o valor dessa variável, mesmo que ela seja uma variável local da função `criar_molde()`.

A função `alternar_estampa()` tem como objetivo alternar entre duas estampas: um caractere passado como argumento para o parâmetro `estampa` e um espaço em branco. Para isso, a função `alternar_estampa()` altera o valor da variável `proxima` com base na comparação entre `estampa` e o valor atual de `proxima`.

Na linha 2, `*proxima` é um operador de desreferência que obtém o valor armazenado na memória apontada pelo ponteiro `proxima`. Um operador de desreferência, representado pelo símbolo `*`, é usado para acessarmos o conteúdo de uma variável apontada por um ponteiro.

Nessa linha, a função `alternar_estampa()` compara o valor apontado por `proxima` com o valor de `estampa`. Se forem iguais, o valor de `proxima` é alterado para um espaço em branco (' '). Se forem diferentes, o valor de `proxima` é alterado para o valor de `estampa`. Por fim, na linha 3, a função retorna o valor que agora está em `proxima`. Com esse procedimento, ora a função retorna o próprio caractere de estampa, ora a função retorna um espaço em branco.

De volta à linha 10, o caractere retornado pela função `alternar_estampa()` é atribuído à posição atual da matriz. Em seguida, o laço for interno acessará a próxima coluna da matriz, invocará novamente a função `alternar_estampa()` para sua atribuição e assim por diante, até que o programa atinja a última coluna da matriz.

Ao sair do laço `for` interno, o programa executa a linha 11, que faz parte do bloco de comandos do laço `for` externo, destinado a percorrer as linhas da matriz. A linha 11 do código é responsável por atualizar o valor da variável `proxima` com o caractere posicionado na coluna 0 da linha atual, antes de o programa passar para a próxima linha da matriz. Desse modo, se, por exemplo, a linha 2 da matriz foi iniciada com o caractere da estampa, a linha 3 será iniciada com o espaço em branco.

A execução da função `criar_molde()` é encerrada apenas após o programa percorrer todas as posições da matriz, fazendo as devidas impressões em cada uma delas.

3. Análise das alternativas

A - Alternativa incorreta.

JUSTIFICATIVA. O código apresentado não gera estampas alternadas para matrizes com número de colunas par, devido à inexistência da linha 11 do código apresentado na resolução da questão, reproduzida novamente a seguir.

```
proxima = molde[i][0];
```

Para um número par de colunas, o preenchimento das linhas sempre se iniciará com a impressão do caractere da estampa, gerando um padrão próximo ao exposto a seguir.

```
# # # #
# # # #
# # # #
```

B - Alternativa incorreta.

JUSTIFICATIVA. Além de apresentar o mesmo problema do código da alternativa A, o segundo parâmetro da função `alternar_estampa()` não foi declarado como um ponteiro. Desse modo, `alternar_estampa()` não é capaz de modificar o conteúdo da variável `proxima`, que é uma variável local da função `criar_molde()`. Para criar estampas alternadas, o endereço da variável `proxima` deve ser passado como argumento para um ponteiro, de modo que as alterações feitas dentro da função `alternar_estampa()` afetem a variável original `proxima`.

C - Alternativa incorreta.

JUSTIFICATIVA. Nesse código, o problema está na atualização da variável `proxima`. A função `alternar_estampa()` retorna o caractere alternado, mas não é feita a atualização do valor da variável `proxima`, por meio do comando de atribuição. Como resultado, todos os elementos na mesma linha receberão a mesma estampa.

D - Alternativa correta.

JUSTIFICATIVA. Ver resolução da questão.

E - Alternativa incorreta.

JUSTIFICATIVA. O segundo parâmetro da função `alternar_estampa()` não foi declarado como um ponteiro. Desse modo, `alternar_estampa()` não é capaz de modificar o conteúdo da variável `proxima`, que é uma variável local da função `criar_molde()`.

3. Indicações bibliográficas

- BACKES, A. *Linguagem C*: completa e descomplicada. Rio de Janeiro: LTC, 2023.
- SOFFNER, R. *Algoritmos e programação em linguagem C*. São Paulo: Saraiva, 2013.
- BACKES, A. R. *Algoritmos e estruturas de dados em linguagem C*. Rio de Janeiro: LTC, 2023.

ÍNDICE REMISSIVO

Questão 1	Operações lógicas. Comandos condicionais. Sistemas especialistas.
Questão 2	Lógica quantitativa. Operações lógicas. Comandos condicionais aninhados. Programação estruturada (linguagem C).
Questão 3	Organização de computadores. Sistemas de numeração. Conversão entre bases numéricas.
Questão 4	Sistemas operacionais. Gerenciamento de memória.
Questão 5	Lógica de programação estruturada. Variáveis indexadas (vetores). Laços de repetição.
Questão 6	Lógica de programação estruturada. Variáveis indexadas (vetores e matrizes). Laços de repetição aninhados.
Questão 7	Programação estruturada (linguagem C). Variáveis indexadas (matrizes). Funções. Ponteiros. Condicional ternária.