UNIP
EDUCAÇÃO, POR SUA ESCOLHA

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 18

CQA/UNIP – Comissão de Qualificação e Avaliação da UNIP

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 18

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.

José Carlos Morilla

Doutor e Mestre em Engenharia de Materiais, Mestre em Engenharia de Produção, Especialista em Engenharia Metalúrgica, Especialista em Física e Engenheiro Mecânico, com MBA em Gestão de Empresas. Professor adjunto da Universidade Paulista.

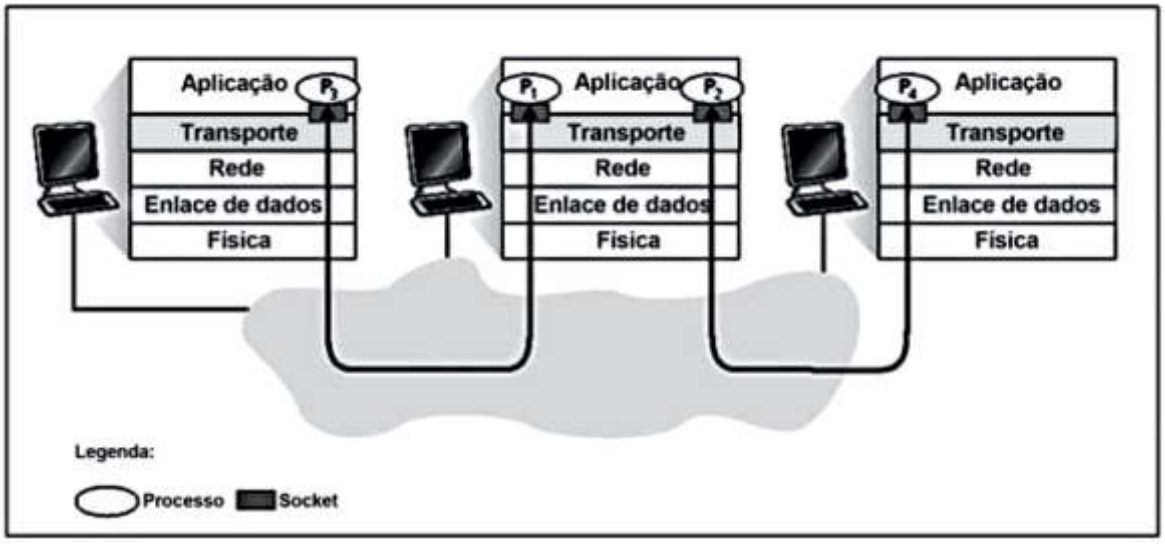
Tiago Guglielmeti Correale

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

Questão 1

Questão 1.[1]

Em uma rede de computadores baseada na arquitetura TCP/IP, a camada de transporte fornece o serviço de comunicação fim a fim entre os processos de aplicações executadas em cada um dos hospedeiros. Do ponto de vista das aplicações, a comunicação lógica fim a fim significa que tudo se passa como se os hospedeiros que rodam os pares de processos estivessem conectados diretamente. A figura a seguir mostra o recorte de uma rede com três hospedeiros e quatro processos. Observe a existência de uma comunicação lógica entre os processos P1 e P3 e entre os processos P2 e P4.



KUROSE, J. F.; ROSS, K. W. Redes de computadores e a internet: uma abordagem top-down. 5. ed. São Paulo: Addison Wesley, 2010 (com adaptações).

Com base na figura apresentada, assinale a opção correta.

A. O processo P3 se conecta a P1 por meio de um número de porta UDP, que é um identificador exclusivo entre 0 e 1023.
B. No hospedeiro do meio, o serviço de multiplexação/demultiplexação fornecido pela camada de transporte garante a entrega dos dados ao processo de aplicação apropriado.
C. O endereço IP de destino contido no cabeçalho do protocolo TCP identifica o hospedeiro para o qual o segmento está sendo enviado e garante a comunicação fim a fim.
D. O protocolo de camada de transporte usado na comunicação entre os pares de processos é o TCP, que utiliza uma apresentação de três vias para estabelecer a comunicação fim a fim.

[1]Questão 11 – Enade 2017.

E. No protocolo de camada de transporte utilizado na comunicação entre os processos P2 e P4, cada segmento de dados enviado entre esses dois processos obterá uma confirmação de recebimento.

1. Análise da questão

Protocolos TCP e UDP

A questão aborda o uso dos protocolos TCP e UDP em um cenário bastante simplificado. Um dos pontos explorados pela questão refere-se aos processos de multiplexação e de demultiplexação. Para compreendermos esses conceitos, podemos imaginar um cenário no qual um computador está executando diversos processos simultaneamente. Suponha que mais de um processo está conectado a outros processos, execute em outros computadores e se comunique via uma rede que utiliza os protocolos TCP/IP. O sistema operacional, por meio dos seus serviços de rede, especificamente da implementação da pilha de protocolos TCP/IP, deve fazer com que cada processo "veja" apenas a sua comunicação, separando os diversos fluxos de dados dos processos executando em paralelo.

A pilha de protocolos TCP/IP, também chamados de protocolos internet, utiliza uma estratégia na qual identificamos cada máquina, ou, mais precisamente, cada interface de rede, por meio de um endereço IP. Mas, além do endereço IP, um processo pode utilizar uma ou mais portas. A associação do endereço IP e da porta permite que vários processos executando em um mesmo computador se comuniquem com outros processos paralelamente.

A ligação entre os processos executando em diferentes máquinas é feita com a ajuda de sockets (traduções possíveis para esse termo seriam "tomada" ou "encaixe", mas, normalmente, utilizamos o vocábulo em inglês). Do ponto de vista do processo, a comunicação é feita pelo envio das informações para o socket, sendo que os demais cuidados são feitos pela pilha de protocolos TCP/IP. Na criação do socket, usamos o endereço IP da máquina que desejamos contactar e a porta associada a determinado serviço ou processo. De forma simplificada, podemos pensar que uma porta é um número de 16 bits utilizado de forma a selecionar o processo ao qual queremos nos conectar. Reforçamos que precisamos de duas informações: o endereço IP e o número da porta.

A ideia de porta permite que, em uma mesma máquina executando vários processos diferentes e simultâneos, possam ser estabelecidos fluxos de comunicação individuais para cada um dos processos. Além disso, todos esses processos vão utilizar a mesma interface de

rede e o mesmo endereço IP (associado a essa interface). Assim, por exemplo, em um mesmo computador, podem ser executados simultaneamente um servidor web na porta 80 e um servidor de ftp na porta 21. Os dois processos têm seus fluxos de dados individuais e separados, pois trabalham em portas diferentes. No entanto, o endereço IP é o mesmo.

Tanto o protocolo UDP quanto o protocolo TCP oferecem um mecanismo de porta (ambos são protocolos da camada de transporte). No caso do protocolo UDP, contudo, a comunicação é feita sem o estabelecimento de conexão, mas utilizando sockets. Vale notar que o protocolo TCP exige o estabelecimento de uma conexão entre os processos e também usa sockets para a comunicação.

2. Análise das alternativas

A - Alternativa incorreta.

JUSTIFICATIVA. Para que o processo P3 se conecte ao processo P1, precisamos utilizar a combinação de um endereço IP e de uma porta. O valor da porta é especificado por um número de 16 bits. Logo, há 2^{16} = 65.536 possibilidades. Assim, o número das portas varia de 0 até 65.535. No entanto, devemos lembrar que as portas mais baixas (de 0 até 1023) são reservadas.

B - Alternativa correta.

JUSTIFICATIVA. Os processos de multiplexação e de demultiplexação permitem que a camada de transporte seja capaz de "coletar" e "distribuir" as diversas informações enviadas aos diversos processos ou recebidas pelos diversos processos que executam simultaneamente em um sistema operacional. Por exemplo, dois programas que executam em uma mesma máquina podem enviar para um servidor na rede ou receber dados de um servidor na rede, sendo que a camada de transporte tem de ser capaz de separar essas informações e entregá-las ao processo correspondente.

C - Alternativa incorreta.

JUSTIFICATIVA. Ainda que o enunciado afirme que a rede utilizada é baseada na arquitetura TCP/IP, não é possível saber se a aplicação usa o protocolo TCP ou o protocolo UDP para a camada de transporte. Caso seja empregado o protocolo UDP, não podemos garantir a entrega dos pacotes. Adicionalmente, mesmo no caso TCP, seriam necessárias outras informações, como a porta do destino.

D - Alternativa incorreta.

JUSTIFICATIVA. Não podemos garantir, pelo contexto do enunciado, que o protocolo da camada de transporte utilizado seja o TCP. Outra possibilidade é o uso do protocolo UDP e, também, da camada de transporte.

E - Alternativa incorreta.

JUSTIFICATIVA. Não sabemos se o protocolo utilizado é o TCP ou o UDP. Se fosse o protocolo UDP, não haveria garantia da entrega de pacotes.

3. Indicações bibliográficas

- KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a Internet: uma abordagem top-down*. 6. ed. São Paulo: Pearson Education do Brasil, 2014.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE. G. *Fundamentos de sistemas operacionais*. 4. ed. Rio de Janeiro: LTC, 2015.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 4. ed. São Paulo: Pearson Education do Brasil, 2016.
- TANENBAUM, A. S.; WETHERALL, J. *Redes de computadores*. 5. ed. São Paulo: Pearson Education do Brasil, 2011.

Questão 2

Questão 2.[2]

O protocolo IPv6 foi desenvolvido para substituir o IPv4, tendo sua implementação ocasionado várias mudanças importantes, como a capacidade de endereçamento expandida, o cabeçalho aprimorado de 40 bytes e a rotulação de fluxo e prioridade.

Considerando essas informações, avalie as afirmativas a seguir, relativas à descrição dos campos de cabeçalho IPv6.

I. Em "endereço de origem" e "endereço de destino", cada campo possui 64 bits, tendo sido expandidos os 32 bits usados no IPv4.

II. Em se tratando do cabeçalho IPv6, insere-se o valor 32 no campo "versão" de 4 bits que é usado para identificar a versão do protocolo IP.

III. O campo "próximo cabeçalho" identifica o protocolo ao qual os dados presentes no datagrama serão entregues, por exemplo, TCP ou UDP.

IV. No IPv6, o campo "classe de tráfego" de 8 bits é semelhante ao campo "tipo de serviço" do IPv4, ambos utilizados para diferenciar os tipos de pacotes IP.

V. O valor do campo "limite de saltos" é decrementado em um para cada roteador que repassa o pacote; caso a contagem do limite de salto chegue a zero, o pacote será descartado.

É correto apenas o que se afirma em

A. II e IV.

B. I, II e III.

C. I, III e V.

D. III, IV e V.

E. I, II, IV e V.

1. Análise da questão

Protocolo IP

O IP é um protocolo da camada de rede fundamental para a construção da Internet e das diversas redes baseadas na pilha de protocolos TCP/IP. Podemos dizer que a versão IPv4

[2]Questão 13 – Enade 2017.

é uma das mais populares. No entanto, tal versão apresenta um problema: a limitação com relação ao seu espaço de endereçamento.

O endereço IP é um número de 32 bits, o que impõe uma restrição ao número máximo de endereços possíveis de aproximadamente 4 bilhões. Em virtude do crescimento de uso e a popularização da Internet, esse número se tornou claramente insuficiente. Com as finalidades de ampliar o espaço de endereçamento disponível e de melhorar diversos aspectos do protocolo, foi criada uma nova versão, o padrão IPv6, que dispõe de um espaço de endereçamento muito maior, além de oferecer as funcionalidades de rotulação de fluxo e de prioridade.

2. Análise das afirmativas

I - Afirmativa incorreta.
JUSTIFICATIVA. No IPv6, os campos "endereço de origem" e "endereço de destino" têm 128 bits de tamanho, e não 64 bits, como indicado no texto da afirmativa.

II - Afirmativa incorreta.
JUSTIFICATIVA. O valor a ser inserido é o número 6 (na forma decimal) ou 0110 (na forma binária), e não 32, como indicado no texto da afirmativa.

III - Afirmativa correta.
JUSTIFICATIVA. Esse campo é normalmente utilizado para indicar qual é o protocolo a ser utilizado pela camada de transporte, ou seja, se é o protocolo TCP ou o protocolo UDP.

IV - Afirmativa correta.
JUSTIFICATIVA. O campo "classe de tráfego" (em inglês, "traffic class") tem duas funções:

- classificação dos pacotes (via o campo "differentiated services");
- identificação de congestionamento.

V - Afirmativa correta.
JUSTIFICATIVA. O objetivo desse campo é limitar a durabilidade dos dados em circulação em uma rede, evitando que pacotes muito antigos (e que provavelmente se tornaram inúteis) fiquem circulando indefinidamente e congestionando a rede de modo desnecessário.

Alternativa correta: D.

3. Indicações bibliográficas

- KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a Internet: uma abordagem top-down*. 6. ed. São Paulo: Pearson Education do Brasil, 2014.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE. G. *Fundamentos de sistemas operacionais*. 4. ed. Rio de Janeiro: LTC, 2015.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 4. ed. São Paulo: Pearson Education do Brasil, 2016.
- TANENBAUM, A. S.; WETHERALL, J. *Redes de computadores*. 5. ed. São Paulo: Pearson Education do Brasil, 2011.

Questão 3

Questão 3.[3]

Em uma rede local de uma instituição bancária, foi implantado um sistema gerenciador de banco de dados (SGDB), tendo o administrador efetuado um comando GRANT para a realização das configurações de acesso ao SGDB na rede.

Com o objetivo de realizar autenticação por aplicações para acesso ao banco de dados, de forma segura, devem ser enviados ao servidor

A. o IP do servidor e a senha.
B. a senha e o endereço MAC.
C. o IP do servidor e o endereço MAC.
D. o IP do servidor, o nome físico do banco e a senha.
E. o IP do cliente, o nome físico do banco, a senha e o endereço MAC.

1. Análise da questão

Sistemas gerenciadores de banco de dados (SGBDs)

Existem diversos tipos de sistemas gerenciadores de banco de dados (SGBDs) no mercado. O tamanho e a complexidade desses sistemas dependem muito do tipo de aplicação a que se destinam.

Alguns sistemas de gerenciamento de banco de dados são extremamente pequenos, como é o caso da pequena biblioteca SQLite em C. Ela é, em geral, integrada à aplicação e pode ser utilizada de diversas formas, como, por exemplo, para armazenar informações de configurações dos usuários de um programa.

Existem também sistemas de gerenciamento de banco de dados grandes e complexos, projetados para atender a milhares de solicitações de muitos usuários simultaneamente, como o Oracle, o Microsoft SQL Server e o Postgres, entre outros. Nesses casos, os SGBDs utilizam tipicamente uma arquitetura cliente-servidor.

Sistemas criados com tais SGBDs normalmente são compostos por um software cliente que se conecta diretamente ao SGBD ou a um servidor de aplicação (e esse último se conecta ao SGBD). Nesse tipo de cenário, é bastante comum que vários clientes diferentes se conectem simultaneamente a um SGBD, enviando consultas e recebendo resultados.

[3]Questão 16 – Enade 2017.

Nos casos da utilização da arquitetura cliente-servidor, a questão da segurança é fundamental. Para garantir um mínimo de segurança na utilização da base de dados, o administrador do SGDB (o DBA, em geral) deve criar um usuário no banco de dados. Esse usuário tem um perfil associado, de forma similar a um usuário de um sistema operacional, com limitações de acordo com as necessidades da aplicação. Normalmente, definimos uma senha para garantir um mínimo de segurança: assim, evitamos que qualquer pessoa com acesso à rede seja capaz de fazer consultas no banco de dados.

Adicionalmente, é possível que uma mesma máquina física tenha várias instâncias de banco de dados diferentes. Ainda que o SGDB seja o mesmo, as instâncias individuais podem pertencer a projetos diferentes, com diferentes nomes (associados na sua criação). A chamada "string de conexão" é o conjunto de informações necessárias para que uma aplicação seja capaz de se conectar a uma instância, o que envolve várias informações, como o nome do banco de dados, o endereço IP do servidor, o nome do usuário e uma senha.

2. Análise das alternativas

A - Alternativa incorreta.
JUSTIFICATIVA. A aplicação deve indicar o nome da instância na qual vai ser feita a conexão.

B - Alternativa incorreta.
JUSTIFICATIVA. Existem várias formas de se conectar a uma base de dados e nem todas elas utilizam o protocolo TCP/IP. Contudo, os casos mais comuns envolvem a identificação do endereço IP do servidor, o nome da base de dados, o nome de usuário e uma senha. A identificação do endereço MAC associado ao endereço IP do servidor é um serviço feito pelos protocolos da rede.

C - Alternativa incorreta.
JUSTIFICATIVA. Os protocolos da rede são responsáveis por identificar o endereço MAC associado ao endereço IP do servidor (quando necessário). Além disso, devemos prover informações como o nome da instância do banco de dados, o usuário e uma senha.

D - Alternativa correta.
JUSTIFICATIVA. O endereço IP é utilizado para identificar a máquina na rede que está com o SGDB instalado. O nome do banco identifica a instância específica à qual se quer conectar.

Além disso, normalmente é necessário que um usuário e uma senha também sejam fornecidos, garantindo um mínimo de segurança para a base de dados.

E - Alternativa incorreta.
JUSTIFICATIVA. Na conexão, deve ser indicado o endereço IP do servidor ao qual se deseja conectar. Além disso, a identificação do endereço MAC associado ao endereço IP do servidor é uma tarefa da rede.

3. Indicações bibliográficas

- KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a Internet: uma abordagem top-down*. 6. ed. São Paulo: Pearson Education do Brasil, 2014.
- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE. G. *Fundamentos de sistemas operacionais*. 4. ed. Rio de Janeiro: LTC, 2015.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de banco de dados*. 5. ed. Rio de Janeiro: Elsevier, 2006.
- TANENBAUM, A. S.; BOS, H. *Sistemas operacionais modernos*. 4. ed. São Paulo: Pearson Education do Brasil, 2016.
- TANENBAUM, A. S.; WETHERALL, J. *Redes de computadores*. 5. ed. São Paulo: Pearson Education do Brasil, 2011.

Questão 4

Questão 4.[4]

O código Java, a seguir, contém a implementação de uma Pilha utilizando a estratégia encadeada.

```
public class No{
      public int dado;
      public No prox;
      public No(int dado){
            this.dado = dado;
      }
}
public class PilhaEncadeada{
      private No topo;
      public int pop(){
            if (topo == null)
                  return -1;
            No lixo = topo;
            topo = topo.prox;
            lixo.prox = null;
            return lixo.dado;
      }
      public int top(){
            if (topo == null)
                  return -1;
            return (topo.dado);
      }
}
```

Com base no exposto, assinale a opção que possui a implementação correta do método push para essa Pilha.

A.
```
public void push(int elemento){
      No novo = new No(elemento);
      if(topo == null)
            topo = novo;
      else
            topo.prox = novo;
}
```

B.
```
public void push(int elemento) {
      No novo = new No(elemento);
      topo = novo;
}
```

C.
```
public void push(int elemento) {
      topo.dado = elemento;
}
```

[4]Questão 11 – Enade 2021.

D.
```
public void push(int elemento) {
        No novo = new No(elemento);
        while(topo != null)
                topo = topo.prox;
        topo.prox = novo;
    }
```
E.
```
public void push(int elemento) {
        No novo = new No(elemento);
        novo.prox = topo;
        topo = novo;
     }
```

1. Análise da questão

A questão aborda uma importante estrutura de dados, a pilha, que é implementada por meio de um código escrito em Java, uma linguagem de programação orientada a objetos. Alguns conceitos importantes para a resolução da questão serão abordados a seguir.

1.1. Pilha

No contexto de estruturas de dados, uma **pilha** é uma forma específica de lista de elementos organizados de acordo com o princípio "Last In, First Out" (LIFO). Essa é uma forma de armazenamento e organização de dados em que o último elemento adicionado é o primeiro a ser removido. É possível inserirmos objetos em uma pilha a qualquer momento, mas somente o objeto inserido mais recentemente (ou seja, o último que "entrou") pode ser removido a qualquer momento.

Para conseguirmos visualizar essa estrutura com mais facilidade, imagine que estamos em um restaurante estilo buffet, onde há uma pilha de pratos disponível para os clientes. Ao se aproximar da pilha, é natural que você, como cliente, pegue o prato do topo para se servir. Quando um funcionário se aproxima com o intuito de repor pratos limpos à pilha, é natural que ele coloque esses novos elementos no topo, sobre os outros pratos já existentes na pilha. Desse modo, o último prato inserido pelo funcionário será o primeiro prato a ser retirado pelo próximo cliente. Essa situação ilustra bem o princípio LIFO da estrutura de dados.

Nesse contexto, uma **pilha encadeada**, mencionada no enunciado da questão, é uma implementação da estrutura de dados pilha que utiliza uma lista encadeada. Nessa estrutura, cada elemento da pilha é chamado de **nó**. Cada nó contém o valor do elemento e uma referência (ou ponteiro) para o próximo nó da pilha.

1.2. Métodos de pilha

Métodos de pilha são operações específicas que podem ser aplicadas a uma estrutura de dados de pilha, como adicionar elementos, remover elementos e verificar o elemento no topo, entre outras.

Formalmente, uma pilha S é um tipo abstrato de dados (TAD) que suporta os dois métodos que seguem.

- push(e): insere o objeto e no topo da pilha.
- pop(): remove o elemento no topo da pilha e o retorna. Ocorre um erro se a pilha estiver vazia.

Em uma pilha encadeada, todas as operações de inserção (push) e remoção (pop) ocorrem no início (topo) da lista ligada. Navegadores de internet, por exemplo, armazenam em uma pilha os endereços recentemente visitados. Toda vez que o navegador visita um novo site, o endereço desse site é armazenado na pilha de endereços (operação push). Utilizando o botão "voltar", o navegador permite que o usuário retorne a sites previamente visitados (operação pop).

Adicionalmente, podem ser definidos os métodos de pilha listados a seguir.

- size(): retorna o número de elementos na pilha.
- isEmpty(): retorna um booleano indicando se a pilha está vazia.
- top(): retorna o elemento no topo da pilha, sem retirá-lo. Ocorre um erro se a pilha estiver vazia.

A tabela 1 mostra uma série de operações de pilha e seus efeitos sobre o conteúdo de uma pilha S de inteiros, inicialmente vazia. Na primeira operação, o método push() insere o elemento 5 na pilha, fazendo com que seu conteúdo seja (5). A segunda operação insere no topo da pilha o valor 3, fazendo com que seu conteúdo seja (5, 3). A terceira operação, que usa o método pop(), remove o elemento do topo da pilha e retorna-o. Com isso, o elemento 3 saiu do conteúdo e foi para a saída. A sequência de operações continua, de forma a exemplificar a ação de cada um dos métodos aplicados à pilha. Pela convenção adotada para a descrição do conteúdo, o elemento de topo está à direita e o elemento de base está à esquerda.

Vejamos a tabela 1.

Tabela 1. Operações sobre uma pilha S, inicialmente vazia.

Operação	**Saída**	**Conteúdo da pilha S na ordem: fim ← S ← topo**
push(5)	-	(5)
push(3)	-	(5, 3)
pop()	3	(5)
push(7)	-	(5, 7)
pop()	7	(5)
top()	5	(5)
pop()	5	()
pop()	"error"	()
isEmpty()	true	()
push(9)	-	(9)
push(7)	-	(9, 7)
push(3)	-	(9, 7, 3)
push(5)	-	(9, 7, 3, 5)
size()	4	(9, 7, 3, 5)
pop()	5	(9, 7, 3)
push(8)	-	(9, 7, 3, 8)
pop()	8	(9, 7, 3)
pop()	3	(9, 7)

GOODRICH, M. T.; TAMASSIA, R. *Estruturas de dados e algoritmos em Java*. Porto Alegre: Bookman, 2013 (com adaptações).

1.3. Análise do código-fonte do enunciado

No código da questão, é feita a implementação de uma pilha encadeada, utilizando a linguagem Java.

A estrutura da pilha é baseada em nós, onde cada nó contém um valor inteiro e uma referência ao próximo nó na pilha. Vamos avaliar dois trechos, sendo o primeiro reproduzido abaixo, que representa a classe No.

```
public class No{
    public int dado;
    public No prox;
    public No(int dado){
        this.dado = dado;
    }
}
```

Nesse trecho, vemos a classe No, que representa a estrutura de um nó da pilha. Essa classe tem dois campos: dado, para armazenar um valor inteiro, e prox, para apontar para o próximo nó da pilha.

Em seguida, é inicializado o campo dado de um nó com o valor passado como argumento ao criar um novo nó. Isso permite que cada nó da pilha armazene um valor inteiro específico, que é fornecido quando o nó é criado.

Vamos, agora, avaliar o trecho de código correspondente à classe PilhaEncadeada, cujo código é reproduzido a seguir.

```
public class PilhaEncadeada{
    private No topo;
    public int pop(){
        if (topo == null)
            return -1;
        No lixo = topo;
        topo = topo.prox;
        lixo.prox = null;
        return lixo.dado;
    }
    public int top(){
        if (topo == null)
            return -1;
        return (topo.dado);
    }
}
```

A classe PilhaEncadeada é a classe principal que representa a pilha encadeada. Ela tem um campo topo, que representa o nó no topo da pilha. Nesse trecho, há a implementação de dois métodos: pop() e top().

O método pop() é usado para remover o elemento no topo da pilha. Ele verifica se a pilha está vazia (se topo for null). Se estiver vazia, ele retorna o valor -1. Se a pilha não estiver vazia, ele remove o nó no topo da pilha, atualiza a referência topo para apontar para o próximo nó e retorna o valor do nó removido. No contexto do código fornecido, a variável lixo representa o nó que está sendo removido da pilha.

O método top() é usado para obter o elemento no topo da pilha sem removê-lo. Ele verifica se a pilha está vazia. Se estiver vazia, ele retorna o valor -1. Se a pilha não estiver vazia, ele retorna o valor do nó do topo da pilha.

Note, no entanto, que o código original não incluiu o método push(), e a implementação correta desse método deve ser encontrada em uma das alternativas fornecidas.

2. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. O trecho da implementação do método push() da alternativa não insere corretamente o novo nó no topo da pilha nem atualiza o campo prox do novo nó. Nesse código, há atualização do campo prox do nó no topo da pilha, mas não há a adequada conexão do novo nó ao restante da pilha. Isso resulta em perda de referência aos outros elementos, o que não pode acontecer em uma pilha encadeada.

B – Alternativa incorreta.

JUSTIFICATIVA. Nesse trecho, há apenas a substituição do topo da pilha pelo novo nó, o que significa que a referência para qualquer nó já existente na pilha é perdida. Isso não é uma implementação válida, já que devemos adicionar elementos no topo da pilha sem perder os elementos anteriores.

C – Alternativa incorreta.

JUSTIFICATIVA. Esse trecho apenas atualiza o valor do dado do nó existente no topo da pilha, em vez de criar um novo nó.

D – Alternativa incorreta.

JUSTIFICATIVA. Esse trecho busca percorrer toda a pilha, movendo topo até que topo seja nulo. Isso faz com que topo perca a referência aos elementos anteriores na pilha, o que os torna inacessíveis.

E – Alternativa correta.

JUSTIFICATIVA. Nessa implementação, foi criado um novo nó, chamado de novo, com o elemento a ser inserido, que foi recebido como argumento. Em seguida, foi definido o campo prox do novo nó para apontar para o nó que, atualmente, está no topo da pilha. Finalmente, o topo foi atualizado para ser o novo nó que acabamos de criar. Com isso, o novo nó se torna o novo topo da pilha.

Esse algoritmo segue uma estratégia adequada para adicionar um elemento à pilha encadeada, inserindo o novo elemento no topo da pilha e mantendo a ordem correta dos elementos.

3. Indicações bibliográficas

- GOODRICH, M. T.; TAMASSIA, R. *Estruturas de dados e algoritmos em Java*. Porto Alegre: Bookman, 2013.
- LAMBERT, K. A. *Fundamentos de Python: estruturas de dados*. São Paulo: Cengage Learning, 2022.
- BACKES, A. R. *Algoritmos e estruturas de dados em linguagem C*. Rio de Janeiro: LTC, 2023.

Questão 5

Questão 5.[5]

Considere que as variáveis pilha e fila correspondem, respectivamente, às estruturas de dados do tipo Pilha e Fila. Para testar as duas estruturas, um programador realizou a série de operações a seguir.

```
Pilha pilha = new Pilha();
Fila fila = new Fila();
pilha.push('A');
pilha.push('B');
pilha.push('C');
fila.enqueue(pilha.top());
fila.enqueue(pilha.top());
fila.enqueue('D');
pilha.push(fila.dequeue());
fila.enqueue(fila.dequeue());
fila.enqueue(pilha.pop());
pilha.push('E');
fila.enqueue('E');
pilha.pop();
```

Após essas operações, ao imprimir o conteúdo de pilha e fila, respectivamente, serão exibidos os seguintes resultados:

A. pilha: topo → C → A → E.
   fila: início → D → A → A → E.

B. pilha: topo →A.
   fila: início → D → B → C → E.

C. pilha: topo → C → B → A.
   fila: início → D → C → C → E.

D. pilha: topo → B → A.
   fila: início → D → B → C → E.

E. pilha: topo → C → B → A.
   fila: início → D → B → C → E.

[5]Questão 13 – Enade 2021.

1. Análise da questão

Métodos de pilha e de fila

No contexto de estruturas de dados, uma **pilha** é uma maneira específica de lista de elementos organizados de acordo com o princípio "Last In, First Out" (LIFO). Essa é uma forma de armazenamento e de organização de dados em que o último elemento adicionado é o primeiro a ser removido. Os principais métodos associados a uma pilha são: o método push(), que permite a inserção de um elemento, e o método pop(), que retira da pilha o último elemento adicionado e o retorna. Além disso, o método top() retorna o elemento presente no topo da pilha, sem removê-lo.

Uma **fila** também constitui uma lista de elementos organizados. No entanto, essa organização se dá pelo princípio "First In, First Out" (FIFO). Assim, nessa forma de organização, o primeiro elemento adicionado, que está na frente da fila, é o primeiro a ser removido. Em uma fila, os elementos podem ser inseridos a qualquer momento, mas somente o elemento que está na fila há mais tempo pode ser retirado em um dado momento.

O tipo abstrato de dados (TAD) fila suporta os dois métodos fundamentais, elencados a seguir.

- enqueue(e): insere o elemento e no fim da fila.
- dequeue( ): retira e retorna o objeto da frente da fila. Ocorrerá um erro se a fila estiver vazia.

Adicionalmente, o TAD fila inclui os métodos auxiliares dispostos a seguir.

- size( ): retorna o número de objetos na fila.
- isEmpty( ): retorna um booleano indicando se a fila está vazia.
- front( ): retorna, mas não remove, o objeto na frente da fila. Ocorre um erro se a fila estiver vazia.

Para exemplificar a ação desses métodos, a tabela 1 mostra uma série de operações e os seus efeitos sobre uma fila Q, inicialmente vazia, de inteiros. Na exibição do conteúdo da fila, o elemento posicionado mais à esquerda está na frente da fila.

Tabela 1. Operações sobre uma fila Q, inicialmente vazia.

Operação	**Conteúdo da fila Q na ordem: início ← Q ← fim**
enqueue(5)	(5)
enqueue(3)	(5, 3)
dequeue()	(3)
enqueue(7)	(3, 7)
dequeue()	(7)
front()	(7)
dequeue()	()
isEmpty()	()
enqueue(9)	(9)
size()	(9)

GOODRICH, M. T.; TAMASSIA, R. *Estruturas de dados e algoritmos em Java*. Porto Alegre: Bookman, 2013 (com adaptações).

2. Resolução da questão

No código fornecido no enunciado da questão, os dois primeiros comandos inicializam uma pilha vazia e uma fila vazia, respectivamente, e armazenam essas estruturas de dados nas variáveis pilha e fila, para uso posterior no código.

A partir daí, os métodos de pilha e de fila começam a ser utilizados. A tabela 2 reúne os efeitos de cada operação sobre as estruturas de dados consideradas, onde S representa a pilha (com elemento de topo à esquerda) e Q representa a fila (com elemento de início à esquerda).

Tabela 2. Efeitos de cada operação sobre as estruturas de dados consideradas.

	Operação	**Pilha topo → S → fim**	**Fila início ← Q ← fim**
1	pilha.push('A');	(A)	()
2	pilha.push('B');	(B, A)	()
3	pilha.push('C');	(C, B, A)	()

4	fila.enqueue(pilha.top());	(C, B, A)	(C)
5	fila.enqueue(pilha.top());	(C, B, A)	(C, C)
6	fila.enqueue('D');	(C, B, A)	(C, C, D)
7	pilha.push(fila.dequeue());	(C, C, B, A)	(C, D)
8	fila.enqueue(fila.dequeue());	(C, C, B, A)	(D, C)
9	fila.enqueue(pilha.pop());	(C, B, A)	(D, C, C)
10	pilha.push('E');	(E, C, B, A)	(D, C, C)
11	fila.enqueue('E');	(E, C, B, A)	(D, C, C, E)
12	pilha.pop();	(C, B, A)	(D, C, C, E)

A seguir, acompanharemos uma breve explicação a respeito de cada operação listada na tabela.

1) O elemento 'A' é adicionado ao topo da pilha. Enquanto isso, a fila continua vazia.
2) O elemento 'B' é adicionado ao topo da pilha. A fila continua vazia.
3) O elemento 'C' é adicionado ao topo da pilha. A fila continua vazia.
4) O comando fila.enqueue(pilha.top()); faz com que o elemento de topo da pilha (que é o elemento 'C') seja adicionado ao final da fila.
5) Novamente, o elemento de topo da pilha (elemento C) é adicionado ao final da fila.
6) O elemento 'D' é adicionado ao final da fila.
7) O comando pilha.push(fila.dequeue()); faz com que, primeiro, seja removido o elemento de início da fila (elemento 'C'). Depois, esse elemento removido é adicionado ao topo da pilha.
8) O comando fila.enqueue(fila.dequeue()); faz com que, primeiro, seja removido o elemento de início da fila (elemento 'C'). Depois, esse elemento é adicionado ao final da fila.
9) O comando fila.enqueue(pilha.pop()); faz com que, primeiro, seja removido o elemento de topo da pilha (elemento 'C'). Depois, esse elemento é adicionado ao final da fila.
10) O elemento 'E' é adicionado ao topo da pilha.
11) O elemento 'E' é adicionado ao final da fila.
12) O elemento de topo da pilha (elemento 'E') é removido.

Ao fim das operações, ao imprimir o conteúdo das estruturas, será apresentado o conteúdo disposto a seguir.

pilha: topo → C → B → A.

fila: início → D → C → C → E.

Alternativa correta: C.

3. Indicações bibliográficas

- GOODRICH, M. T.; TAMASSIA, R. *Estruturas de dados e algoritmos em Java*. Porto Alegre: Bookman, 2013.
- LAMBERT, K. A. *Fundamentos de Python*: estruturas de dados. São Paulo: Cengage Learning, 2022.
- BACKES, A. R. *Algoritmos e estruturas de dados em linguagem C*. Rio de Janeiro: LTC, 2023.

Questão 6

Questão 6.[6]

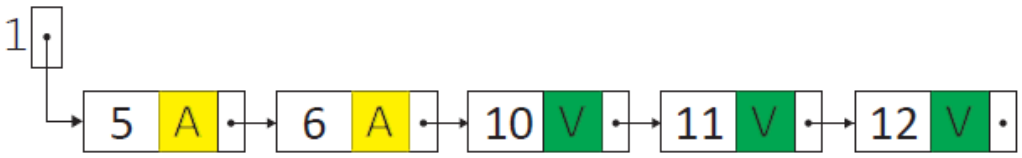
Com a finalidade de melhorar o atendimento e priorizar os casos mais urgentes, a direção de um hospital criou um sistema de triagem em que um profissional da saúde classifica a ordem de atendimento com base numa avaliação prévia do paciente, entregando-lhe um cartão numerado verde (V) ou amarelo (A), que define o menor ou maior grau de urgência da ocorrência, respectivamente. Para informatizar esse processo, o software desenvolvido tem como base o trecho de código-fonte a seguir.

```
1    struct lista {
2         int numero;
3         char cor;
4         struct lista* prox;
5    };
6
7    typedef struct lista Lista;
8
9    Lista* inserir_fim(Lista* l, Lista* no) {
10        Lista* aux;
11
12        no->prox = NULL;
13        aux = l;
14        while (aux->prox != NULL)
15             aux = aux->prox;
16        aux->prox = no;
17
18        return l;
19   }
20
21   Lista* inserir(Lista* l, int numero, char cor) {
22        Lista* no = (Lista*) malloc(sizeof(Lista));
23        no->numero = numero;
24        no->cor = cor;
25
26        if (l == NULL) {
27             no->prox = l;
28             l = no;
29        } else {
30             if (no->cor == 'V')
31                  l = inserir_fim(l, no);
32             else
33                  l = inserir_prioridade(l, no);
34        }
```

[6]Questão Discursiva 5 – Enade 2021.

```
35
36          return l;
37    }
```

Na linha 21, a função inserir recebe o número e a cor do cartão entregue ao paciente na triagem. Pacientes com cartão verde são inseridos no final da fila pela função inserir_fim (linhas 9-19). Pacientes com cartão amarelo têm prioridade no atendimento e são inseridos no início da fila, em ordem de chegada, pela função inserir_prioridade. Portanto, se são entregues os cartões 10-V, 11-V, 5-A, 12-V e 6-A, nessa ordem, a fila deve ficar assim organizada:



Considerando o processo de triagem descrito e o trecho de código-fonte apresentado, implemente a função inserir_prioridade conforme indicado.

```
Lista* inserir_prioridade(Lista* l, Lista* no)
```

1. Análise da questão

A questão testa os conhecimentos a respeito de estruturas de dados, como a estrutura heterogênea struct e a lista simplesmente encadeada. Além disso, também são verificados conhecimentos sobre o uso de ponteiros, implementados por meio da linguagem C.

1.1. Estrutura heterogênea struct

Dependendo da situação que desejamos modelar em nosso programa estruturado, as variáveis definidas com os tipos de dados primitivos e as estruturas homogêneas (arrays) podem não ser suficientes. A linguagem C permite que o programador crie suas estruturas a partir dos tipos de dados básicos. Para isso, o comando struct pode ser utilizado.

Um **struct** (também chamado de **registro**) pode ser visto como um agrupamento de variáveis sob o mesmo nome, em que cada uma delas pode ter qualquer tipo (inclusive o tipo de outra estrutura). Estamos, desse modo, criando uma estrutura heterogênea. Uma estrutura heterogênea, portanto, é uma estrutura de dados cujos campos podem ser compostos por

variáveis de tipos distintos. Um vetor, por exemplo, pode ser considerado uma estrutura homogênea, já que organiza dados do mesmo tipo, necessariamente.

A forma geral de um struct é apresentada a seguir.

```
struct nome_struct {
        tipo1 campo1;
        tipo2 campo2;
        ...
};
```

Como exemplo, vamos declarar um struct de três campos, para representar os dados de cadastro de uma pessoa. Esse struct é mostrado a seguir.

```
1    struct cadastro {
2          char nome[100];
3          int idade;
4          char endereço[100];
5    };
```

Note que os campos do struct são declarados da mesma forma que as variáveis primitivas ou arrays. Além disso, os nomes dos campos de um struct devem ser diferentes uns dos outros. Nesse exemplo, o nome do struct é cadastro, que servirá como um novo "tipo de dado". Com ele, podemos declarar novas variáveis, do tipo struct cadastro.

Uma vez definido o "esqueleto" do struct, conforme acabamos de fazer, as variáveis estruturadas podem ser declaradas de modo similar aos tipos de dados já existentes. Por exemplo, para declararmos o cadastro da Maria e do Alex, podemos usar o comando a seguir.

```
struct cadastro maria, alex;
```

Nesse caso, maria é uma variável estruturada do tipo struct cadastro, que servirá para abrigar os dados da Maria. O mesmo vale para alex: é uma variável estruturada do tipo struct cadastro, mas que abrigará os dados do Alex.

Uma vez definida uma variável do tipo do struct, é preciso poder acessar seus campos para se trabalhar. Cada campo do struct pode ser acessado pelo operador "." (ponto), conforme a sintaxe exposta a seguir.

```
variavel.campo
```

Por exemplo, para acessarmos o campo idade da variável estruturada maria, podemos usar, em nosso programa, o comando maria.idade.

Podemos passar variáveis estruturadas como argumento para uma função por valor ou por referência. Quando passamos uma variável estruturada **por valor**, ocorre uma cópia dos dados da estrutura original, o que tem dois aspectos importantes: primeiro, mudanças "na cópia" não vão ser refletidas na estrutura original; segundo, há uma penalidade em desempenho (por causa do tempo para a cópia dos dados) e um aumento do consumo de memória (por causa da duplicação dos dados em memória). Para passarmos uma variável estruturada por valor, temos duas possibilidades: passar todos os campos da variável estruturada ou passar apenas alguns campos. As duas possibilidades são elencadas a seguir.

- Passar uma **variável estruturada** por valor: para isso, basta declararmos, na lista de parâmetros, um parâmetro com o mesmo tipo da estrutura heterogênea. Dessa forma, teremos acesso a todos os campos da estrutura dentro da função.
- Para passar **um campo** de uma variável estruturada por valor: basta declararmos, na lista de parâmetros, um parâmetro com o mesmo tipo do campo a ser passado.

Quando precisamos que a função realize alterações em campos de uma variável estruturada local de outra função, podemos passar os campos ou a própria variável estruturada **por referência**. Nesse caso, passamos o endereço para um ponteiro, que será um parâmetro da função em questão. As duas possibilidades são elencadas a seguir.

- Passar uma **variável estruturada** por referência: para passarmos o endereço de uma variável estruturada, o parâmetro da função deverá ser um ponteiro do tipo da estrutura heterogênea. Após a passagem, para usar a notação de ponto com o ponteiro, precisamos indicá-lo entre parênteses no código. Por exemplo: para acessar o campo `x` de uma variável estruturada a partir de um ponteiro `p` e atribuirmos o valor 10 a esse campo, fazemos:

  ```
  (*p).x = 10;
  ```

  Podemos, também, substituir o par de parênteses e os operadores de asterisco e de ponto apenas pelo operador `->`. O comando a seguir é análogo ao comando anterior.

  ```
  p->x = 10;
  ```

- Para passar **um campo** de uma variável estruturada por referência: devemos passar o endereço do campo e, nesse caso, o parâmetro da função deverá ser um ponteiro compatível com o tipo do campo cujo endereço ele vai receber. O campo será tratado como um ponteiro para uma variável convencional.

1.2. Lista simplesmente encadeada

No contexto da ciência da computação, uma **lista** é uma estrutura de dados linear utilizada para armazenar e organizar dados em um sistema computacional. Uma estrutura do tipo lista, no seu uso mais comum, é uma sequência de elementos do mesmo tipo. Existem várias representações diferentes para uma lista. Essas representações variam a depender de como os elementos são inseridos ou removidos da lista, do tipo de alocação usada e do tipo de acesso aos elementos. As estruturas pilha e fila, por exemplo, podem ser facilmente implementadas utilizando, como base, uma estrutura de lista encadeada simples.

As listas podem ser implementadas por algoritmos estruturados, utilizando recursos como ponteiros e comandos struct.

Uma lista encadeada, em sua forma mais simples, é uma estrutura de dados na qual os elementos são chamados de **nós** (ou nodos). Os nós, juntos, formam uma ordem. Cada nó é uma estrutura que armazena um elemento e uma referência, que chamaremos de prox, para outro nó. Uma lista encadeada geralmente é implementada por mecanismos de alocação dinâmica e, nesse caso, ela não tem um tamanho fixo predeterminado e usa um espaço proporcional à quantidade de seus elementos. Além disso, os nós de uma lista encadeada não são indexados.

Uma **lista linear simplesmente encadeada** é um tipo de lista em que os nós formam uma estrutura linear. O último nó da lista possui um ponteiro nulo (que aponta para NULL), ou seja, um ponteiro que não aponta para qualquer endereço de memória válido, indicando o final da lista.

Essa estrutura pode ser ilustrada conforme mostrado na figura 1. Podemos ver, nessa figura, a existência de um ponteiro de nome L, que aponta para o primeiro nó da lista. Esse primeiro nó contém um valor (Info 1) e um ponteiro prox (representado por uma seta) para o próximo nó. O segundo nó contém um valor (Info 2) e um ponteiro prox para o próximo nó. O terceiro e último nó contém um valor (Info 3) e um ponteiro prox que aponta para NULL (representado por /), indicando que a lista linear simplesmente encadeada atingiu seu fim.

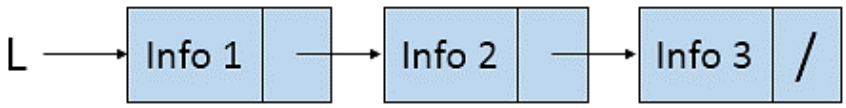


Figura 1. Estrutura de uma lista linear simples encadeada. Os ponteiros prox de cada nó são representados como setas e o valor NULL é representado por /.

Portanto, a estrutura da lista é construída de forma encadeada, onde cada nó traz uma referência ao próximo nó. Isso permite o acesso sequencial aos elementos da lista, começando pelo primeiro nó (ou início da lista) e percorrendo os nós subsequentes, até chegar ao final da lista.

Uma lista simplesmente encadeada mantém seus elementos em determinada ordem, que pode depender da forma como os elementos são inseridos na lista. Essa ordem é definida pela cadeia de ligações prox que parte de um nó para seu sucessor na lista. Como os nós de uma lista linear simplesmente encadeada não são indexados, apenas examinando um nó individual, não podemos dizer se ele é o segundo, o terceiro ou o décimo nó da lista.

1.3. Análise do código-fonte do enunciado

No código da questão, a estrutura heterogênea struct lista representa a estrutura dos nós individuais de uma lista simplesmente encadeada, onde cada nó contém um número e uma cor como informações. Cada nó, portanto, representa um cartão. O campo prox é um ponteiro que aponta para o próximo nó na lista.

Vamos entender os trechos do código-fonte apresentado no enunciado. Leve em consideração o trecho a seguir.

```
1    struct lista {
2         int numero;
3         char cor;
4         struct lista* prox;
5    };
```

O trecho acima representa a declaração de uma estrutura heterogênea, de nome lista. A estrutura heterogênea lista contém os campos elencados a seguir.

- numero: variável do tipo int (inteiro), que serve para armazenar o número do cartão.
- cor: variável do tipo char (caractere), com o intuito de armazenar um caractere que representa a cor do cartão. Pelo contexto, deverá assumir o valor verde ('V') ou o valor amarelo ('A'), indicando a prioridade do atendimento.
- prox: ponteiro para uma estrutura do tipo lista, que aponta para o próximo elemento da lista. No contexto, esse ponteiro deve apontar para o cartão do próximo paciente a ser atendido, ou ser igual a NULL (marcando o fim da lista).

Desse modo, essa estrutura é utilizada para representar um nó em uma lista encadeada, em que cada elemento contém um número e uma cor, além de um ponteiro para

o próximo elemento da lista. Com esse método, é possível percorrer os elementos da lista encadeada, ligando-os por meio do ponteiro prox.

Na linha 7, foi utilizado o comando typedef para definir um novo tipo de dado, conforme exposto a seguir.

```
7     typedef struct lista Lista;
```

Nessa linha, o programador está utilizando a definição da estrutura lista, criada nas linhas de 1 a 5, e criando um novo tipo de dado de nome Lista, por meio do comando typedef. Isso torna o código mais conveniente, pois, para declarar variáveis do tipo da estrutura, em vez de escrever o comando struct lista nome_da_variavel, podemos apenas escrever Lista nome_da_variavel.

Vamos, agora, analisar o conteúdo da função descrita entre as linhas 9 e 19, dispostas a seguir.

```
9    Lista* inserir_fim(Lista* l, Lista* no) {
10        Lista* aux;
11
12        no->prox = NULL;
13        aux = l;
14        while (aux->prox != NULL)
15             aux = aux->prox;
16        aux->prox = no;
17
18        return l;
19   }
```

Nesse trecho, é criada uma função chamada inserir_fim(), contendo dois parâmetros: um ponteiro para o primeiro elemento da lista, l, e um ponteiro para o nó a ser inserido, no. Essa função tem como objetivo inserir o nó no ao final da lista simplesmente encadeada. Por meio da função inserir_fim(), pacientes com cartão verde são inseridos no final da fila.

Na linha 10, é declarado o ponteiro aux, que será utilizado para percorrer a lista. Na linha 12, o campo prox do nó no é definido como NULL, indicando que esse será o último elemento da lista. Na linha 13, o ponteiro aux recebe o valor do ponteiro l, que aponta para o primeiro elemento da lista nesse momento.

Entre as linhas 14 e 15, ocorre um laço while que percorre a lista até encontrar o último elemento, identificado pelo campo prox igual a NULL. Durante a execução do laço, o ponteiro aux é atualizado para apontar para o próximo elemento da lista.

Na linha 16, o ponteiro prox do último elemento da lista (aux) é atualizado para apontar para o nó no, inserindo-o no final da lista. Por fim, na linha 18, a função retorna o ponteiro l, que representa o início da lista atualizada, após a inserção do nó no.

De forma resumida, a função inserir_fim() percorre a lista até o último elemento e insere um novo nó no final, ajustando os ponteiros prox apropriados.

É importante notar que, no código-fonte apresentado, inserir_fim() é invocada por um comando existente dentro de outra função personalizada, que é a inserir(). Vamos, então, analisar o conteúdo disposto entre as linhas 21 e 37, cujo trecho de código é apresentado a seguir, relativo à declaração dessa função.

```
21  Lista* inserir(Lista* l, int numero, char cor) {
22      Lista* no = (Lista*) malloc(sizeof(Lista));
23      no->numero = numero;
24      no->cor = cor;
25
26      if (l == NULL) {
27          no->prox = l;
28          l = no;
29      } else {
30          if (no->cor == 'V')
31              l = inserir_fim(l, no);
32          else
33              l = inserir_prioridade(l, no);
34      }
35
36      return l;
37  }
```

Nesse trecho, temos a declaração de uma função inserir(), cujo retorno é do tipo Lista*, que é um ponteiro para um nó, mais especificamente o primeiro nó de uma lista encadeada. Essa função contém três parâmetros: um ponteiro para o início da lista, l, um número inteiro, numero, e um caractere, cor. O objetivo dessa função é criar um novo nó com os valores fornecidos e inseri-lo na lista encadeada, de acordo com determinadas condições, expressas pelos comandos if-else.

Na linha 22, um ponteiro de nome no é criado utilizando a função malloc, para reservar um espaço na memória primária do tamanho da estrutura heterogênea Lista, já que o papel do comando sizeof(Lista) é identificar o tamanho, em bytes, do tipo de dados Lista. A função malloc retorna um ponteiro genérico e, portanto, é necessário fazer um typecast para atribuí-lo ao tipo correto. Esse typecast é feito da seguinte forma: (Lista*).

Na linha 23, o campo numero do nó no é atribuído com o valor do parâmetro numero. Na linha 24, o campo cor do nó no é atribuído com o valor do parâmetro cor.

O caso especial "inserção em uma lista vazia" é tratado de forma separada pelo programa, nas linhas 26-29. Como nesse caso de lista vazia não há outros cartões para estabelecer uma prioridade, a função inserir() associa diretamente o nó criado com o nó inicial da lista, que vai ser retornado na linha 36, e não são chamadas nenhuma das outras funções auxiliares. Isso faz com que as funções inserir_fim() e inserir_prioridade() não se "preocupem" com o caso da lista vazia, e ambas dependem, implicitamente, que as listas não estejam vazias para o seu correto funcionamento.

Na linha 26, é verificado se a lista está vazia. Caso seja verdadeiro (ou seja, se l for igual a NULL), a lista está vazia e o nó no será o primeiro elemento da lista. Caso contrário, a lista já possui pelo menos um nó (potencialmente vários).

As linhas 27 e 28 serão executadas caso a lista esteja vazia. Na linha 27, o campo prox do nó no é atribuído com o valor de l (ou seja, valor NULL). Na linha 28, o ponteiro l é atualizado para apontar para o novo nó no, o que o torna o novo nó inicial da lista.

Na linha 29, temos o comando else, cujo bloco de comandos será executado caso seja falso que a lista está vazia, ou seja, se a lista possui ao menos um elemento. Seu bloco ocorre entre as linhas 30 e 33. Se a lista não estiver vazia, é necessário disparar o mecanismo de inserção correto para cada caso, como estipulado pelo enunciado.

Na linha 30, é verificado se a cor do novo nó no é igual a 'V' que, nesse contexto, significa cor verde. Se a proposição for verdadeira, significa que o novo nó deve ser inserido no final da lista, já que pacientes cujo cartão tem cor verde não são prioritários no atendimento. Isso é feito invocando a função inserir_fim(), que já foi comentada anteriormente.

Naturalmente, caso a proposição na linha 30 seja falsa, isso significa que a cor do novo nó é diferente de 'V', o que, nesse contexto, significa que o paciente tem um cartão amarelo. Nesse caso, o novo nó deve ser inserido em uma posição prioritária na lista. A função inserir_prioridade() é chamada na linha 33, passando l e no como argumentos, e o retorno da função é atribuído a l. A função inserir_prioridade() ainda não foi comentada porque sua implementação é, justamente, o objetivo que temos para resolver a questão, conforme o enunciado. Veremos em detalhes a sua implementação no próximo item.

Por fim, na linha 36 do código-fonte, o ponteiro l, que representa o início da lista, é retornado. O programa do enunciado manipula a lista mantendo sempre um ponteiro para o seu primeiro nó (ou, se a lista estiver vazia, esse ponteiro deve ser igual a NULL), no caso,

identificado pela variável l. Assim, se ocorrer alguma alteração no primeiro elemento da lista, essa variável deve ser atualizada. Evidentemente, se um elemento for inserido ou retirado em um outro ponto diferente, esse ponteiro não deve se alterar. Mas as funções de alteração da lista (como funções de inserção e remoção) devem sempre receber esse ponteiro, e retorná-lo, mesmo que não ocorra alteração.

De forma resumida, a função inserir cria um novo nó com os valores fornecidos como argumentos e o insere na lista encadeada, de acordo com as condições dispostas a seguir.

- Se a lista estiver vazia, o novo nó se torna o início da lista.
- Se a cor do novo nó for 'V' (verde), o novo nó é inserido no final da lista por meio da função inserir_fim().
- Caso a cor do novo nó seja diferente de 'V', o novo nó é inserido em uma posição prioritária na lista, por meio da função inserir_prioridade().

A função inserir_prioridade() é, justamente, a que deve ser implementada na resposta da questão, e pode ser conferida a seguir.

2. Padrão de resposta do INEP

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

> *O respondente deve implementar a função inserir_prioridade() no código fonte apresentado. Assim, a lógica do algoritmo deve atender a dois fundamentos básicos de inserção de nós em uma lista simplesmente encadeada.*
> *1) INSERÇÃO NO INÍCIO: caso ainda não haja nenhum cartão amarelo na fila, o novo nó é inserido no início, como primeiro elemento da fila;*
> *2) INSERÇÃO NO MEIO: se já houver cartões amarelos na fila, o novo nó é inserido no meio ou no final da fila, a depender, respectivamente, da presença ou da ausência de algum cartão verde; para ambos os casos, contudo, a lógica de inserção do nó é exatamente a mesma, pois as variáveis ant e aux são usadas para percorrer a lista simultaneamente e atualizar as referências (ponteiros) entre os nós.*
> *É natural que o estudante siga essa sequência lógica e, portanto, espera-se que escreva a função.*

```
1  Lista* inserir_prioridade(Lista* l, Lista* no) {
2      if (l->cor == 'V') {
3          no->prox = l;
4          l = no;
5      } else {
6          Lista* ant = l;
7          Lista* aux = ant->prox;
8          while (aux != NULL && aux->cor == 'A') {
9              ant = aux;
```

```
10          aux = aux->prox;
11          }
12      ant->prox = no;
13      no->prox = aux;
14  }
15
16  return l;
17  }
```

Observação. Serão aceitas respostas em outra linguagem de programação compatível ou pseudocódigo.

É importante salientar que o padrão de resposta original do INEP indicava, em seu trecho de código, o termo "cartao" para se referir ao campo cor. Como a struct lista não possui nenhum campo "cartao", isso acarretaria um erro de compilação. Isso foi corrigido no trecho disposto anteriormente.

De forma geral, o código da função inserir_prioridade() lida com a inserção de pacientes com cartão amarelo na fila, garantindo que eles sejam inseridos prioritariamente.

A função recebe dois parâmetros: a lista l, que representa o primeiro paciente da espera já existente, e o nó no, que contém as informações do novo paciente a ser inserido na lista.

Uma verificação condicional é feita para determinar se o primeiro paciente da lista recebida tem um cartão verde. Se sim, isso significa que ele deve passar a ser o próximo paciente, e que o paciente a ser inserido atualmente tem prioridade, e passará a ser o primeiro da espera. Isso ocorre porque, se o primeiro elemento da lista tiver um cartão verde, e, sabendo que os cartões amarelos devem ter prioridade sobre os cartões verdes, então podemos concluir que a lista não deve possuir nenhum nó com o campo "cor" associado à cor amarela (pois, se fosse o caso, esse nó deveria vir antes dos demais). Nesse caso, e levando em consideração que estamos nos referindo à inserção de um nó associado à cor amarela, podemos afirmar que o novo nó a ser inserido deve ser o primeiro nó da lista.

Se o resultado da verificação condicionar for falso, pacientes com cartão amarelo devem ser inseridos em ordem de chegada. Para fazer isso, duas variáveis, ant (anterior) e aux (auxiliar), são inicializadas para apontar para o primeiro paciente da lista e para o próximo paciente da lista, respectivamente.

Um laço while é usado para percorrer a fila enquanto o próximo paciente (aux) não for nulo (ou seja, enquanto houver mais pacientes na fila) e tiver um cartão amarelo ('A'). Dentro do laço, as variáveis ant e aux são atualizadas para avançar na fila.

Após sair do laço, o novo paciente no é inserido entre o paciente anterior (ant) e o paciente atual (aux) na fila. Para fazer isso, o ponteiro prox do paciente anterior

(comando ant->prox) é ajustado para apontar para o novo paciente no, e o ponteiro prox do novo paciente no é ajustado para apontar para o paciente atual (aux).

Por fim, a função inserir_prioridade() retorna o ponteiro l atualizado. Esse mesmo ponteiro é então repassado para a função inserir(), que, por sua vez, vai repassar para o programa que está manipulando a lista.

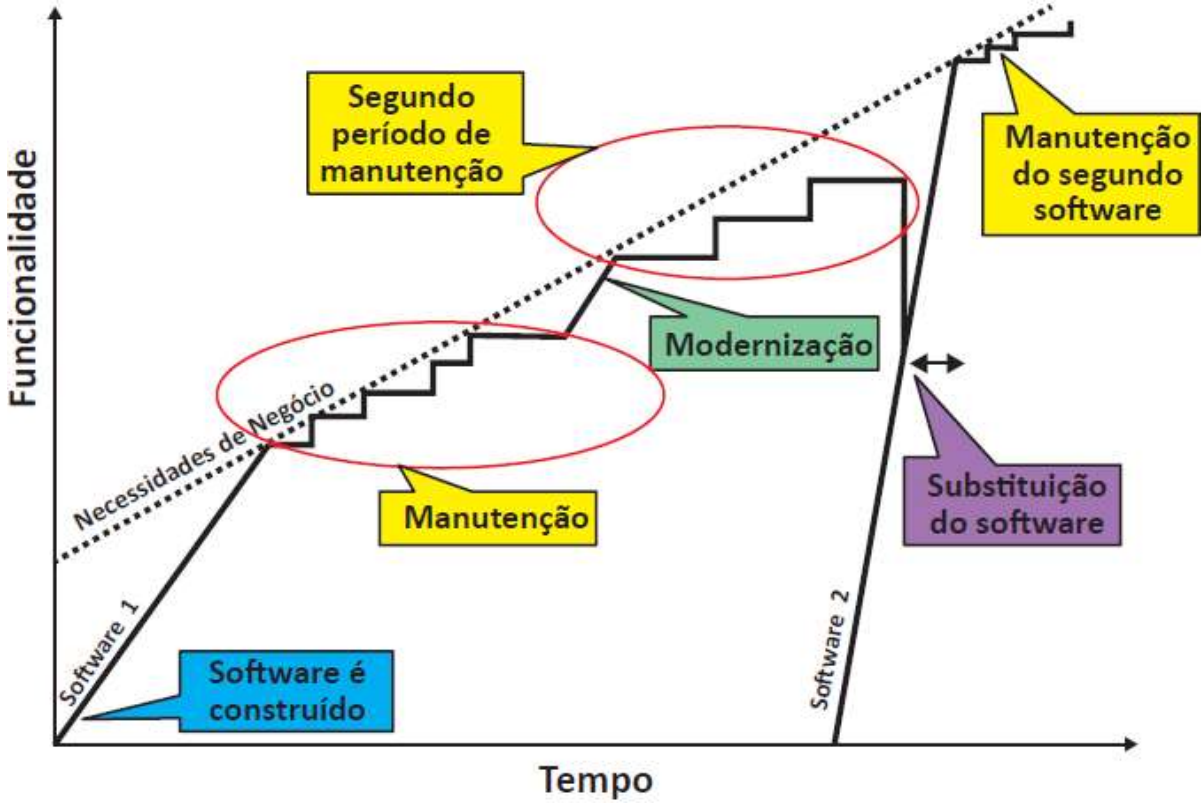
3. Indicações bibliográficas

- 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.
- UNIVERSIDADE FEDERAL DE UBERLÂNDIA. *Revisão de listas simplesmente encadeadas*. Disponível em <https://www.facom.ufu.br/~claudio/Cursos/Antigos/EDxxxx/Artigos/bcc-ed01.pdf>. Acesso em 16 ago. 2025.
- GOODRICH, M. T.; TAMASSIA, R. *Estruturas de dados e algoritmos em Java*. Porto Alegre: Bookman, 2013.
- BACKES, A. *Linguagem C: completa e descomplicada*. Rio de Janeiro: LTC, 2023.
- LAMBERT, K. A. *Fundamentos de Python: estruturas de dados*. São Paulo: Cengage Learning, 2022.

Questão 7

Questão 7.[7]

A evolução de sistemas de software legados pode ser dividida em três categorias: manutenção, modernização e substituição. De acordo com a figura a seguir, conforme o tempo aumenta, a quantidade de funcionalidades também cresce. No primeiro período de evolução a manutenção é realizada, pois as necessidades de negócio são alteradas e o sistema precisa suprir as mudanças por meio da manutenção. No segundo período é realizada a modernização do sistema, pois as necessidades de negócio continuam a crescer, mas há mudanças mais significativas que se fazem necessárias. Além da substituição do sistema de software, nos casos em que o modelo de negócio não atende mais a necessidade da empresa, a manutenção e modernização também não são mais suficientes.



SEACORD, R. C.; PLAKOSH, D.; LEWIS, G. A. *Modernizing Legacy Systems*. Boston: Pearson Education, 2003 (com adaptações).

De acordo com a figura apresentada e considerando um sistema de software implantado de ERP (Enterprise Resource Planning - Planejamento de Recursos Empresariais) contendo um conjunto de módulos que integra todos os departamentos existentes de uma empresa, observou-se a necessidade de criação de um relatório gerencial de comissão da equipe de vendas.

[7]Questão 23 – Enade 2021.

Nesse contexto, é correto afirmar que a manutenção a ser realizada é

A. corretiva.
B. evolutiva.
C. funcional.
D. preventiva.
E. adaptativa.

1. Introdução teórica

Tipos de manutenção de software

Sistemas de software legados são produtos em obsolescência que, normalmente, estão sendo utilizados por uma companhia há muitos anos. No geral, eles são desenvolvidos para ter um longo ciclo de vida. Naturalmente, eles requerem modificações ao longo do tempo.

Nesse contexto, a **manutenção** de um sistema de software legado consiste nos processos de atualização, de correção e de prevenção de problemas em um sistema de software, com o intuito de manter o software operacional no ambiente do usuário. As atividades de manutenção são tipicamente gerenciadas pelo processo de gestão de configuração de software (SCM), que administra as alterações do sistema ao longo de todo o seu ciclo de vida. Existem quatro tipos principais de manutenção de software, elencados a seguir.

- **Corretiva**: este tipo de manutenção é realizado para corrigir erros existentes no software, que não foram identificados ao longo do processo de testes. A correção de bugs, de erros de lógica e de problemas de desempenho são exemplos de manutenções corretivas.
- **Adaptativa**: esta manutenção tem como intuito adaptar o software a novas condições de trabalho, como inseri-lo em um novo ambiente ou fazê-lo funcionar sob novos requisitos de sistema. É exemplo de manutenções adaptativas a tarefa de tornar um software compatível com um novo sistema operacional ou com um novo hardware.
- **Perfectiva (ou evolutiva)**: a manutenção perfectiva tem como foco a adição de novas funcionalidades ao sistema, assim como a melhoria das funcionalidades existentes. Isso inclui a implementação de novos requisitos ou de novas funcionalidades, que não estavam previstas no projeto original.

- **Preventiva**: este tipo de manutenção tem como principal objetivo melhorar a confiabilidade do sistema. Esse tipo de manutenção procura tratar situações que poderão causar erros no software, de maneira a evitar que problemas aconteçam no futuro.

Em média, 50% do tempo dos desenvolvedores costuma ser dedicado à manutenção perfectiva, enquanto as manutenções adaptativa, corretiva e preventiva ocupam, respectivamente, 25%, 21% e 4%. Podemos observar essa distribuição de tempo no gráfico da figura 1.

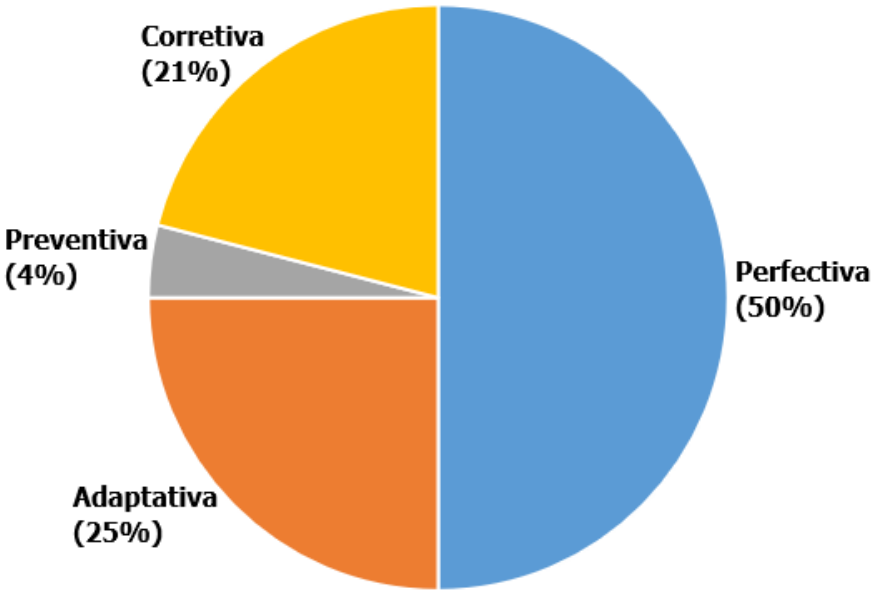


Figura 1. Distribuição do esforço de manutenção.
PRESSMAN e MAXIM, 2021 (com adaptações).

2. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. A manutenção corretiva é realizada para corrigir erros existentes no software. Esse não é o caso da situação explicitada no enunciado.

B – Alternativa correta.

JUSTIFICATIVA. A manutenção evolutiva (ou perfectiva) é realizada quando há necessidade de adicionar novas funcionalidades ao sistema de software, ou melhorar as funcionalidades já existentes. No contexto da questão, a criação de um relatório gerencial é considerada como a adição de uma nova funcionalidade.

C – Alternativa incorreta.

JUSTIFICATIVA. O termo "funcional" não é um termo padrão no contexto da manutenção de software.

D – Alternativa incorreta.

JUSTIFICATIVA. A manutenção preventiva é realizada para evitar futuros problemas, não para adicionar funcionalidades.

E – Alternativa incorreta.

JUSTIFICATIVA. A manutenção adaptativa é realizada para adaptar o software a novos ambientes ou a novos requisitos de sistema. Seu foco não é adicionar funcionalidades, mas sim adaptar as funcionalidades existentes a um novo cenário.

3. Indicações bibliográficas

- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.

Questão 8

Questão 8.[8]

Leia o texto a seguir.

Um erro comum que se observa nas abordagens da Internet das Coisas (Internet of things – IoT) é o desenvolvimento de um produto sem uma visão clara de geração de valor. É comum os desenvolvedores não terem a clareza do real valor a ser gerado aos usuários finais, concebendo um produto sem o seu valor agregado.
Nesse sentido, para abertura de negócios inovadores, é essencial compreender o surgimento de novos negócios baseados em IoT e como essa revolução criará oportunidades.
Disponível em <http://www.convergenciadigital.com.br>. Acesso em 27 jul. 2017 (com adaptações)

Com base nesse contexto, avalie as afirmativas.

I. Para que a IoT possa gerar valor aos negócios, é necessário considerar questões relacionadas a problemas da interoperabilidade de plataformas.
II. O IoT permite uma independência da infraestrutura de telecomunicação e os seus projetos exigem o uso de novas tecnologias, como as impressoras 3D.
III. A tríade composta pela criação de novos mercados digitais, gestão de risco e melhorias de eficiência é a chave para a transformação de modelos de negócios atuais a partir da revolução da IoT.

É correto o que se afirma em

A. I, apenas. B. II, apenas. C. I e III, apenas. D. II e III, apenas. E. I, II e III.

1. Análise da questão

Internet das Coisas *(Internet of things – IoT)*

Desde a década de 1980, temos observado dois fenômenos nas áreas da computação e da eletrônica:

- o aumento do poder computacional;
- a diminuição dos custos de produção.

A produção de equipamentos eletrônicos se tornou cada vez mais barata, ao mesmo tempo em que as possibilidades de aumento na complexidade desses equipamentos se tornaram cada vez maiores.

Outro fenômeno que também verificamos refere-se à explosão, em popularidade, da Internet, vinda da disseminação do uso das redes de computadores.

[8]Questão 33 – Enade 2017.

O efeito integrado de todos esses fenômenos foi o surgimento da possibilidade de fazermos equipamentos eletrônicos que apresentam, além de elevado grau de complexidade computacional (incluindo a utilização da inteligência artificial), a possibilidade de comunicação autônoma em uma rede de computadores. Esse é o mundo da Internet das Coisas, ou IoT (Internet of Things). Contudo, ainda que exista ampla disponibilidade de produção de dispositivos inteligentes e interconectados, não podemos esquecer a dimensão comercial envolvida nos projetos desses produtos.

Nesse contexto, a questão em análise busca determinar se o estudante tem a percepção da agregação de valor que a IoT pode proporcionar aos produtos e aos serviços oferecidos por uma empresa. As afirmativas tratam, também, da dependência existente entre a IoT e a infraestrutura disponível.

Para responder à questão, é importante que o aluno tenha conhecimentos sobre a IoT e sobre como ela pode agregar valor aos produtos e aos serviços oferecidos. Trata-se de um diferencial de sucesso para uma empresa.

2. Análise das afirmativas

I - Afirmativa correta.

JUSTIFICATIVA. Toda a cadeia de desenvolvimento de IoT é suportada pelo hardware, pela rede, pela infraestrutura, pelas plataformas e pelos sistemas integrados. Dessa forma, para que a IoT possa agregar valor aos negócios, é preciso considerarmos questões que viabilizem a interoperabilidade completa em vários níveis, desde o estabelecimento de protocolos e dispositivos até as tarefas que envolvem instâncias do topo da hierarquia institucional.

II - Afirmativa incorreta.

JUSTIFICATIVA. Uma solução de negócios fundamentada em IoT pode ser baseada na infraestrutura de telecomunicações existente na empresa. Isso não requer, necessariamente, o uso de impressoras 3D.

III - Afirmativa correta.

JUSTIFICATIVA. Segundo Cossini (2020),

> *a tríade composta pela criação de novos mercados digitais, gestão de risco e melhorias de eficiência é a chave para a transformação dos modelos de negócio atuais a partir da revolução que a Internet das Coisas vem trazendo para o*

> *bem-estar das pessoas ao redor do planeta. As empresas devem inovar por meio de novas parcerias e uso intensivo de inovadoras tecnologias para levarem produtos e serviços que melhor atendam os novos consumidores digitais e aqueles que, não tão novos, já estão inseridos na economia digital por meio de seus smartphones ou dispositivos inteligentes.*

Alternativa correta: C.

3. Indicações bibliográficas

- BANCO NACIONAL DE DESENVOLVIMENTO ECONÔMICO E SOCIAL – BNDES. *Internet das coisas – Um plano de ação para o Brasil.* Disponível em <http://www.mctic.gov.br/mctic/export/sites/institucional/tecnologia/SEPOD/politicasDigitais/arquivos/arquivos_estudo_iot/fase-4-3.pdf>. Acesso em 18 fev. 2020.
- BANCO NACIONAL DE DESENVOLVIMENTO ECONÔMICO E SOCIAL – BNDES. *Internet das coisas: estimando impactos na economia.* Disponível em <https://www.bndes.gov.br/wps/portal/site/home/conhecimento/noticias/noticia/internet-coisas-iot>. Acesso em 18 fev. 2020.
- COELHO, P. *Internet das Coisas – Introdução prática.* Lisboa: FCA, 2017.
- COSSINI, R. *A Internet das Coisas pode mudar a economia?* Disponível em <https://computerworld.com.br/2016/04/19/internet-das-coisas-pode-mudar-economia/>. Acesso em 18 fev. 2020.
- FURTADO, F.; PEIXOTO, E.; PEREIRA, D. *Roadmap de Educação para IoT-Internet of Things.* Disponível em <https://www.cesar.org.br/index.php/2017/12/12/r oadmap-de-educacao-para-iot%E2%80%8A-%E2%80%8Ainternet-of-things/>. Acesso em 18 fev. 2020.
- MAGRANI, E. *A Internet das Coisas.* Rio de Janeiro, FGV. 2018.
- SINCLAIR, B. *IoT: Como usar a "Internet das Coisas" para alavancar seus negócios.* São Paulo: Autêntica Business, 2018.

ÍNDICE REMISSIVO

Questão 1	Protocolos TCP e UDP.
Questão 2	Protocolo IP.
Questão 3	Sistemas gerenciadores de banco de dados (SGBDs).
Questão 4	Estruturas de dados. Pilha encadeada. Programação orientada a objetos (linguagem Java).
Questão 5	Estruturas de dados. Pilha e fila. Lógica de programação.
Questão 6	Estruturas de dados. Lista simplesmente encadeada. Programação estruturada (linguagem C). Comando struct. Ponteiros.
Questão 7	Engenharia de software. Manutenção de software. Manutenções corretiva, adaptativa, perfectiva e preventiva.
Questão 8	IoT. Inovação.