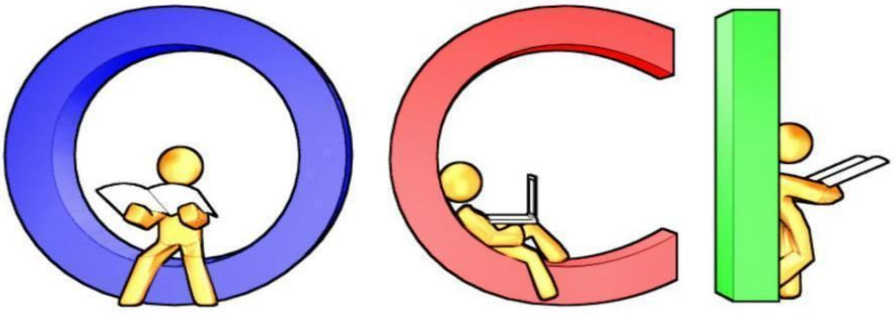


VII Olimpíada Cearense de Informática

2ª FASE

MODALIDADE PROGRAMAÇÃO

Leia atentamente as instruções:

- Não serão permitidos empréstimos de materiais, consultas e comunicação entre os candidatos, tampouco o uso de livros e apontamentos. Relógios e aparelhos eletrônicos em geral deverão ser desligados. O não cumprimento destas exigências ocasionará a exclusão do candidato deste Exame;
- Confira se os dados impressos no Cartão-Resposta correspondem aos seus. Caso haja alguma irregularidade, comunique-a imediatamente ao Aplicador da Prova;
- Aguarde o Aplicador da Prova autorizar a abertura do Caderno de Prova. Após a autorização, confira todas as questões antes de iniciar o Exame;
- Este Caderno de Prova contém 25 (vinte e cinco) questões objetivas, cada qual com apenas 1 (uma) alternativa correta; No Cartão-Resposta, preencha completamente, com caneta de tinta azul ou preta, o retângulo correspondente à alternativa que julgar correta para cada questão;
- No Cartão-Resposta, anulam a questão: a marcação de mais de uma alternativa em uma mesma questão; as rasuras e o preenchimento além dos limites do retângulo destinado para cada marcação. Não haverá substituição do Cartão-Resposta em nenhum desses casos;
- Não serão permitidas perguntas ao Aplicador da Prova sobre as questões da Prova;
- A duração desta prova será de 4 (quatro) horas;
- O tempo mínimo para ausentar-se definitivamente da sala é de 1 (uma) hora;
- Ao concluir a prova, permaneça em seu lugar e comunique ao Aplicador de Prova, sinalizando com uma de suas mãos;
- Aguarde autorização para devolver o Caderno de Prova e o Cartão-Resposta assinado.

Observações

- Nas questões com algoritmos não será utilizada nenhuma linguagem de programação conhecida, e sim pseudocódigo;
- Considere que a primeira posição de um vetor é 0 (zero);
- A função `tamanho()` retorna o tamanho de uma *string* ou vetor;
- A função `imprima(x)` apenas imprime a *string* `x` na tela, enquanto que a função `imprima_pl(x)` imprime a *string* `x` e pula para a próxima linha;
- Caso seja passado um valor não-inteiro (y) para uma variável inteira (x), apenas a parte inteira de y será atribuída à x. Exemplo:
    - `x ← 5,912 // neste caso x recebe apenas 5`

QUESTÃO 01. Considere que num computador de 32 bits o barramento de endereço comporta palavras de 32 dígitos 0 ou 1, logo o tamanho da memória deste computador corresponde ao número de palavras possíveis de serem formadas. Considere também que 1M = 2^{20}, 1G = 2^{30} e 1T = 2^{40}. Um computador de 32 bits é capaz de endereçar quantos bytes de memória?

(A) 4GB;

(B) 2GB;

(C) 4MB;

(D) 2MB;

(E) 0,5TB.

QUESTÃO 02. Sabrina é estudante de Computação e deseja participar da reunião semanal do clube dos matemáticos, que ocorre em uma sala na biblioteca. Todavia, para passar pela porta da sala é necessário fornecer uma senha numérica. Acima da trava da porta está um bilhete:

“A senha dessa semana é o resultado de 100011010010, caso você faça as seguintes considerações:

- A = Adição;
- B = Potenciação;
- C = Subtração;
- D = Divisão;

- E = Multiplicação;
- F = Radiciação.”

Sabrina, muito esperta, teve a ideia de converter o número em binário para hexadecimal e assim conseguiu entrar na reunião.
Logo, Sabrina digitou o número:

(A) 10;

(B) 64;

(C) 6;

(D) 4;

(E) 7.

QUESTÃO 03. Vivemos em um mundo digital e mal conhecemos seus fundamentos. A portas lógicas são a base para qualquer computador ou equipamento avançado. Entender essas portas lógicas nos possibilita entender o avanço da tecnologia atual.

Ao estudar sistema binário, um estudante de computação começou a aprender o funcionamento dessas portas lógicas. Ele viu que a porta OR é aquela que retorna 1 quando ao menos uma de suas entradas é 1. Aprendeu que a porta NOT é aquela que inverte o bit de entrada e que a porta NOR é a porta OR negada, ou seja, se a saída de OR seria 1, a saída equivalente de NOR é 0. Ao abrir o livro viu uma nova nomenclatura, chamada XOR, que a definição era : “A saída de uma porta XOR é determinada pela quantidade de bits 1 de entrada, caso seja um valor ímpar, o valor de saída será 1, e 0 caso seja par.”. Sendo ele curioso, continuou lendo, viu a definição da porta XNOR e um exemplo simples onde uma porta XNOR recebe duas entradas 1 e retorna uma saída X. Levando em conta isso e seus conhecimento prévio sobre portas lógicas, marque o item correto.

(A) A saída X não pode ser calculada pois essa porta não foi feita para esse tipo de cálculo;

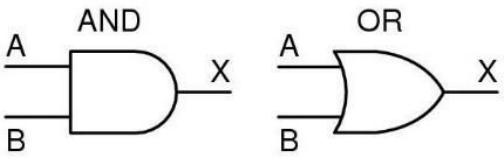
(B) A saída X seria 0;

(C) A saída X seria 1;

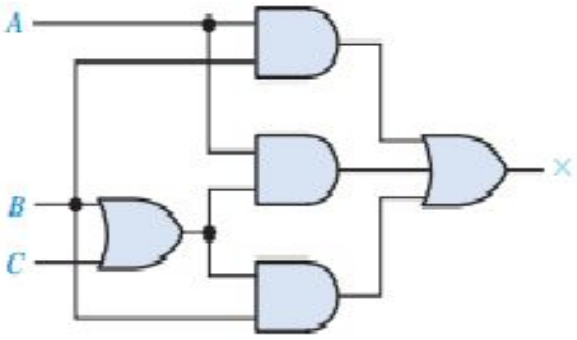
(D) A saída X seria 2;

(E) Ocorreria que não poderia ser feita esse cálculo, apenas caso as entradas fossem (1 e 0), (0, 0) e (0 e 1).

QUESTÃO 04. Pela Álgebra Booleana conseguimos descrever os circuitos que podem construídos pela combinação de portas lógicas. Portanto, nesse tipo de álgebra as variáveis e funções podem ter apenas valores 0 e 1. Sabendo que a saída da porta AND se comporta como a soma de suas entradas e a saída da porta OR como a multiplicação de suas entradas, resolva o problema descrito na questão, tendo como exemplos de portas lógicas e suas tabelas verdades representadas abaixo:



Considerando o circuito abaixo:



Qual a expressão Booleana representa esse circuito lógico?

(A) AB + A(B+C) + B(B+C);

(B) AB + BA(B+C);

(C) (A+B)(A+BC)(B+BC);

(D) AB+BC + AC + A;

(E) AB + AC + BB.

QUESTÃO 05. (Questão adaptada da OBI) O índice de complexidade de um rio é calculado da seguinte maneira:

1. Se o rio não tem afluentes, o seu índice de complexidade é 1;
2. Se o rio tem afluentes de índices diferentes, então o seu índice de complexidade é igual ao do afluente de maior índice;
3. Se todos os afluentes do rio têm o mesmo índice H de complexidade (incluindo o caso quando há apenas um afluente), o seu índice será H+1.

Por exemplo, o rio principal da figura abaixo (representado por uma linha mais grossa) tem índice de complexidade 2.

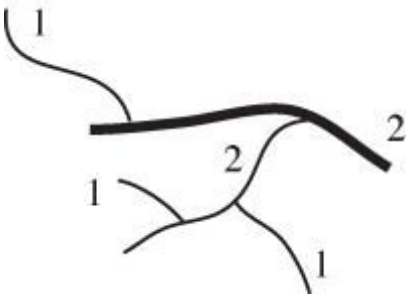


Figura 1 - OBI

Observação: O rio principal ou o afluente principal terá uma linha mais grossa.

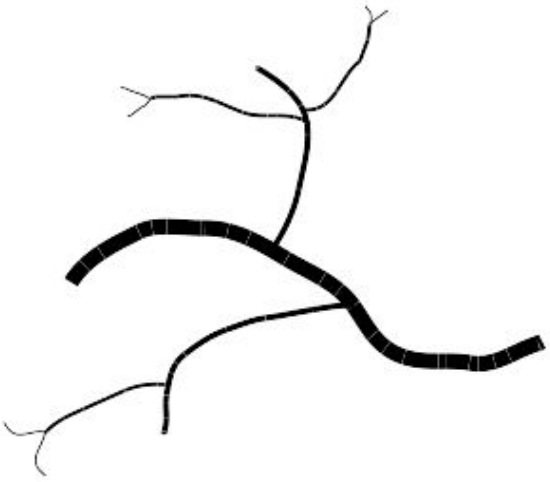


Figura 2 - Arquivo pessoal.

Indique o índice de complexidade para o rio da figura 2 acima.

(A) 3;

(B) 2;

(C) 4;

(D) 5;

(E) 6.

QUESTÃO 06. Alberto, um grande professor de Química e Física Moderna, no auge de sua sabedoria, no intuito de ajudar os alunos com bom raciocínio lógico, propôs que para cada questão da prova de química haveria uma pergunta teórica e um código, onde se o aluno não soubesse com precisão a resposta da questão teórica poderia apelar para o resultado do código para saber a resposta. Em uma dessas provas existia um código do tipo:

```
Programa:
01.  Principal
02.      HS ← 3600
03.      imprima_pl("Entre com um número: ")
04.      leia(X)
05.      H ← (X / HS)
06.      M ← (X - (HS * H)) / 60
07.      S ← (X - (HS*H) - (M*60))
08.      imprima_pl("O valor de H, M, S é:  " + H + "," + M + "," + S)
09.
10.  fim_Principal
```

Onde o X digitado pelo usuário é 3605. A tabela das possíveis respostas da questão teórica para os valores de H, M, S são mostrados nos itens abaixo. Qual item contém os valores corretos de H, M e S?

(A) H = 10, M = 2, S = 0;

(B) H = 10, M = 2, S = 1.;

(C) H =1, M = 0, S = 0.;

(D) H = 1, M = 0, S = 5.;

(E) H = 1, M = 5, S = 0.

QUESTÃO 07. O termo Programação Procedural é um paradigma de programação baseado no conceito de chamadas a procedimento, as quais descrevem um conjunto de passos computacionais a serem executados. Um dado procedimento pode ser chamado a qualquer hora durante a execução de um programa, inclusive por outros procedimentos ou por si mesmo. Ao chamar um procedimento, você pode enviar a ele dados como variáveis, por exemplo. Você tanto pode enviar o endereço da memória onde está armazenada a variável (passagem por referência) como pode enviar uma cópia dessa variável (passagem por valor). Uma passagem por referência faz com que a manipulação dessa variável no seu procedimento altere diretamente o valor que está armazenado no endereço, evitando a necessidade do uso de variáveis globais e otimizando o uso da memória disponível. Já a passagem por valor cria um novo espaço na memória para armazenar o valor passado, ou seja, é feita uma cópia da variável e todas as manipulações realizadas nessa memória serão locais, sendo descartadas ao fim do procedimento.

Veja só um exemplo dessa teoria no pseudocódigo comentado na próxima página:

```
Programa:
01.   passagemReferencia(a)
02.         /* O parâmetro esperado é a referência de uma variável,
logo o valor
03.         recebido é um ponteiro contendo o endereço de memória da
mesma */
04.
05.         a^ ← 0 //Alteração do valor contido no endereço
referenciado pelo ponteiro
06.   fim_passagemReferencia
07.
08.   passagemValor(b)
09.         /*O parâmetro esperado é o valor da variável através de
uma cópia, logo
10.         qualquer operação com a mesma será apenas manipulação
local*/
11.
12.         b ← 1 //Alteração do valor da cópia criada com o valor
passado
13.   fim_passagemValor
14.
```

```
15.   Principal //Início do procedimento principal
16.
17.         leia(x) //Captura de entrada de inteiro inserido pelo
usuário
18.         passagemReferencia(>x) /* Passagem do endereço da variável
x para o
19.         procedimento */
20.         passagemValor(x) //Passagem da variável x para o
procedimento citado
21.         imprima(x)
22.         /*O valor de x é mostrado na tela e este será 0 devido a
operação do
23.         procedimento passagemReferencia()*/
24.
25.   fim_Principal //Fim do procedimento principal
```

Agora que você viu uma aplicação de procedimentos em pseudocódigo e unindo o conhecimento adquirido na questão anterior de ponteiros, responda o que será apresentado na tela como saída com o código a seguir:

```
Programa:
01.   x
02.   p
03.
04.   passagemReferencia(a)
05.         a^ ← (a^) * 7
06.   fim_passagemReferencia
07.
08.   passagemValor(b)
09.         b ← b
10.   fim_passagemValor
11.
12.   Principal
13.         x ← 11
14.         p ← >x
15.         p^ ← p^ % 2
16.
17.         se (x = 0)
18.               passagemValor(x)
19.         senão
20.               passagemReferencia(>x)
21.         fim_se
```

```
22.
23.         imprima(x)
24.   fim_Principal
```

(A) 7;

(B) 1;

(C) 2;

(D) 0;

(E) 14.

QUESTÃO 08. Pseudocódigo é uma forma genérica de se escrever um algoritmo, ou seja, não se leva em consideração as particularidades de cada linguagem. Veja o algoritmo abaixo, escrito em pseudocódigo:

```
Programa:
01.   Principal
02.         pos_par ← 0
03.         pos_impar ← 0
04.         par_continue ← VERDADEIRO
05.         impar_continue ← VERDADEIRO
06.         v_par[5]
07.         v_impar[5]
08.         enquanto (par_continue OU impar_continue)
09.               se (pos_par = 4)
10.                     par_continue ← FALSO
11.               se (pos_impar = 4)
12.                     impar_continue ← FALSO
13.               leia(x)
14.               se ((x % 2) = 0)
15.                     v_par[pos_par] ← x
16.                     se (pos_par = 4)
17.                           pos_par ← 0
18.                     senão
19.                           pos_par ← pos_par + 1
20.               senão
21.                     v_impar[pos_impar] ← x
```

```
22.                     se (pos_impar = 4)
23.                           pos_impar ← 0
24.                     senão
25.                           pos_impar ← pos_impar + 1
26.         fim_enquanto
27.         imprima(v_par[0])
28.         imprima(v_impar[0])
29.   fim_Principal
```

Ao rodar o programa acima, o usuário coloca, a medida que é solicitado pelo programa, as seguintes entradas: 3, 23, 30, 19, 11, 12, 15, 22, 9, 8, 5, 1, 4. Sabendo disso, quais números são esperados como primeira e segunda saída do programa, respectivamente?

(A) 9 e 30;

(B) 4 e 1;

(C) 3 e 30;

(D) 30 e 9;

(E) 30 e 3.

QUESTÃO 09. Em programação quando queremos saber o resto da divisão de um número por outro usamos o operador %, sabendo disso qual o resultado do pseudocódigo abaixo:

```
01.   Principal
02.         A ← 1
03.         B ← 3
04.         SOMA ← 0
05.
06.         para i ← 2 até 5 com passo 1
07.               se i % 2 = 0
08.                     SOMA ← SOMA + A
09.               senão
10.                     SOMA ← SOMA + B
11.               A ← A + 1
12.               B ← B + 1
13.         fim_para
```

```
14.
15.         imprima(SOMA)
16.   fim_Principal
```

(A) 12;

(B) 14;

(C) 15;

(D) 20;

(E) 22.

QUESTÃO 10. Analise o pseudocódigo abaixo e marque o item que demonstra o que será impresso na tela:

```
Programa:
01.   Principal
02.         leia(A)
03.         leia(B)
04.         X ← A
05.         Y ← B
06.         RESTO ← -1
07.         enquanto RESTO ≠ 0
08.               RESTO ← X % Y
09.               X ← Y
10.               Y ← RESTO
11.         fim_enquanto
12.         K ← (A * B) / X
13.         imprima(K)
14.   fim_Principal
```

(A) Imprime o resto da divisão entre A e B;

(B) Imprime o máximo divisor comum entre A e B;

(C) Imprime o máximo divisor comum entre A e B;

(D) Imprime o resultado do produto de A e B ao quadrado;

(E) Imprime o resultado da fatoração de A e B.

QUESTÃO 11. Observe o algoritmo abaixo (considere que a Matriz é inicializada com zeros, e que o primeiro índice das linhas e colunas também é zero):

```
Programa:
01.   Principal
02.         matriz[5][5]
03.
04.         ini ← 0
05.         subQ ← 5
06.
07.         enquanto (subQ > 0)
08.               para i ← ini até 4 - ini com passo 1
09.                     para j ← ini até 4 - ini com passo 1
10.                           matriz[i][j] ← matriz[i][j] + 1
11.                     fim_para
12.               fim_para
13.
14.               ini ← ini + 1
15.               subQ ← subQ - 2
16.         fim_enquanto
17.
```

18. fim_Principal

Após o fim desse algoritmo, qual será o conteúdo da matriz?

(A)
```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

(B)
```
1 2 3 4 5

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

(C)
```
1 1 1 1 1

1 2 2 2 1
1 2 3 2 1
1 2 2 2 1
1 1 1 1 1
```

(D)
```
1 2 3 4 5

2 1 2 3 4
3 2 1 2 3
4 3 2 1 2
5 4 3 2 1
```

(E)
```
3 3 3 3 3

3 2 2 2 3
3 2 1 2 3
3 2 2 2 3
3 3 3 3 3
```

QUESTÃO 12. Torre de Hanói é um desafio em que consiste em uma base com três pinos. No primeiro pino, há ‘n’ discos ordenados por ordem crescente de tamanho começando de cima para baixo. Ou seja, o **maior** e **último** disco está **embaixo** enquanto o **menor** e **primeiro** disco está **em cima**. O objetivo do desafio é deixar um dos dois pinos vazios com a configuração igual ao primeiro, usando o restante como auxiliar na movimentação dos discos. Entretanto, os discos discos só podem ser movidos com as seguintes restrições:

1. Não pode-se mover mais que um disco por vez;
2. Não é permitido colocar um disco de maior diâmetro em cima de outro com diâmetro menor.

Joãozinho, para tentar resolver esse problema, fez o seguinte programa para lhe dizer quais os movimentos necessários para completar o desafio:

Programa:

```
01. Mover_Discos(Qnt_disco, origem, destino, auxiliar)
02.   se (Qnt_disco = 1)
03.         imprima_pl(“mova disco ” + Qnt_disco + “ do ” + origem + “ ao ” + destino)
04.   senão
05.         Mover_Discos(Qnt_disco - 1, origem, auxiliar, destino)
06.         imprima_pl(“mova disco ” + Qnt_disco + “ do ” + origem + “ ao ” + destino)
07.         Mover_Discos(Qnt_disco - 1, auxiliar, destino, origem)
08. fim_Mover_Discos
09.
10. Principal
11.         leia(Qnt_disco)
12.         Mover_Discos(Qnt_disco, primeiro_pino, terceiro_pino, segundo_pino)
13. fim_Principal
```

Para o desafio, Joãozinho fez o teste com **3 (três) discos**. Em relação ao teste, temos que:

(A) O teste deu erro, pois o programa não tem linhas para imprimir os movimentos necessários;

(B) O teste deu erro, pois o programa está incorreto;

(C) O teste foi bem sucedido e Joãozinho precisou fazer 16 (dezesseis) movimentos para completá-lo;

(D) O teste foi bem sucedido e o pino que ficou na mesma configuração do primeiro foi o segundo;

(E) O teste foi bem sucedido e Joãozinho precisou fazer 7 (sete) movimentos para completá-lo.

QUESTÃO 13. O professor Rômulo ensinou a seus alunos sobre os números triangulares, dizendo que os números triangulares são o produto de três números consecutivos. Por exemplo, o número 24 é um número triangular, pois é o resultado da multiplicação dos números 2, 3 e 4. Sabendo disso, o aluno Rhanielzinho fez um código para dizer se um número é triangular ou não.

```
Programa:
01.   Triangular(numero)
02.         a ← 1
03.         b ← 2
04.         c ← 3
05.         enquanto (    )
06.               se (numero = (a * b * c))
07.                   retorne verdadeiro
08.               fim_se
09.               a++
10.               b++
11.               c++
12.         fim_enquanto
13.         retorne falso
14.   fim_Triangular
```

Sabendo que o código de Rhanielzinho está correto, qual seria uma boa condição para o laço “enquanto”?

(A) numero ≠ verdadeiro;

(B) c < numero;

(C) `a * b * c ≤ numero;`

(D) `(a * b) < c;`

(E) `numero = verdadeiro.`

QUESTÃO 14. Durante a aula de programação, o professor solicitou a turma que criassem um algoritmo para o cálculo de números fatoriais. Um determinado aluno desenvolveu o algoritmo abaixo:

```
Programa:
01.  Fatorial(num)
02.    fat ← 1
03.    se num = 0
04.      retorne fat
05.    fim_se
06.    enquanto (num > 1)
07.      fat ← fat * num
08.      num ← num - 1
09.    fim_enquanto
10.    retorne fat
11.  fim_Fatorial
12.
13.  Principal:
14.    int num
15.    imprima("Digite um número positivo: ")
16.    leia(num)
17.    imprima("O fatorial desse número é: ", fatorial(num))
```

```
18.  fim_Principal
```

Observação: A variável **fat** possui 32 bits.
Ao executar o algoritmo, o aluno percebeu que para alguns valores o algoritmo funcionava corretamente, mas à medida que ele tentava calcular o fatorial de números maiores, os resultados não coincidiam com a realidade.
Durante um desses testes, o aluno tentou calcular o fatorial de 13, tendo como retorno do seu algoritmo 1.932.053.504, mas ao fazer o mesmo cálculo usando a calculadora de seu computador o resultado era 6.227.020.800.
Frente a isso, o aluno resolveu estudar para corrigir seu código e descobriu que o erro em questão era:

(A) Estouro de memória;

(B) Erro de Sintaxe;

(C) Warning;

(D) Exceção;

(E) Bug do Compilador.

QUESTÃO 15. Um aluno, obcecado por modularização em seus códigos, decidiu implementar várias funções para facilitar sua vida na resolução de equações polinomiais monovariáveis. Para tal, ele implementou um código onde se deve chamar as funções responsáveis por cada operação necessária para a representação da equação desejada. Inicialmente, ele criou apenas as funções responsáveis por exponenciação, multiplicação, adição e subtração (`Exp(a, b)`, `Mul(a, b)`, `Som(a, b)` e `Sub(a, b)`, respectivamente).

Apesar dos grandes esforços em tentar realizar tal proeza, ele implementou de uma forma equivocada a função **`Exp`**`(a, b)`, responsável por realizar a exponenciação de uma base **a** pelo seu expoente **b**, ambos informados como parâmetros da função.

Segue a implementação de seu código abaixo:

Programa:

```
01.   Exp(a, b)
02.      se (b > 0)
03.         retorne a * Exp(a, b - 1)
```

```
04.      senão
05.         retorne a
06.      fim_se
07.   fim_Exp
08.
09.   Mul(a, b)
10.      retorne a * b
11.   fim_Mul
12.
13.   Som(a, b)
14.      retorne a + b
15.   fim_Som
16.
17.   Sub(a, b)
18.      retorne a - b
19.   fim_Sub
20.
21.   Principal
22.      leia(c)
23.                                          equação          ←
Som(Som(Sub(Exp(c,3),Mul(4,Exp(c,2))),Mul(3,c)),10)
24.   fim_Principal
```

Analisando o código acima, podemos concluir que a expressão polinomial que representa a variável equação é:

(A) $c^4 - 4 * c^3 + 3 * c + 10$;

(B) $c^3 - 4 * c^2 + 3 * c + 10$;

(C) $c^2 - 4 * c^3 + 3 * c + 10$;

(D) $c^4 + 4 * c^3 + 3 * c - 10$;

(E) $c^2 + 4 * c + 3 * c + 10$.

QUESTÃO 16. Suponha que você está vendendo um software de controle de compras de passagens, em uma empresa de transporte rodoviário, e precisa explicar ao dono da empresa o funcionamento do código.

Sabe-se que, em funcionamento normal, o cliente informa se deseja a poltrona no corredor ou na janela (24 lugares no corredor e 24 lugares na janela). Depois, o programa informa as poltronas disponíveis. Caso a escolha não esteja disponível, ou ônibus esteja lotado, uma mensagem deve ser exibida.

Segue o algoritmo:

```
Programa:
01. Principal
02.     corredor[24], janela[24]
03.     para i ← 0 até 23 com passo 1
04.        corredor [i] ← 0
05.        janela[i] ← 0
06.     fim_para
07.     leia(posicao)
08.     achou ← FALSO
09.     para i ← 0 até 23 com passo 1
10.         se (corredor[i] = 0) ou (janela[i] = 0)
11.            achou ← VERDADEIRO
12.         fim_se
13.     fim_para
14.     se (achou = FALSO)
15.         imprima_pl(“Ônibus lotado”)
16.     senão
17.         se (posicao = “Janela”)
18.            achou ← FALSO
19.            para i ← 0 até 23 com passo 1
20.                se (janela[i] = 0)
21.                    imprima_pl(i)
22.                    achou ← VERDADEIRO
23.                fim_se
24.            fim_para
25.         fim_se
26.         se (achou = FALSO)
27.            imprima_pl(“Não tem lugar disponível na janela”)
28.         fim_se
29.         se (posicao = “Corredor”)
30.            achou ← FALSO
31.            para i ← 0 até 23 com passo 1
32.                se (corredor[i] = 0)
33.                  imprima_pl(i)
34.                  achou ← VERDADEIRO
35.                fim_se
36.            fim_para
37.         fim_se
38.         se (achou = FALSO)
39.            imprima_pl (“Não tem lugar disponível no corredor”)
40.         fim_se
41.     fim_se
42. fim_Principal
```

Considerando o algoritmo e a sua função analise as alternativas:

(A)

```
03.    para i ← 0 até 23 com passo 1
04.       corredor [i] ← 0
05.       janela[i] ← 0
06.    fim_para
```

Esta parte do código realiza a inicialização dos dois vetores, corredor[i] e janale[i], atribuindo zero a todos os seu elementos. Trata-se de uma etapa irrelevante do processo, podendo ser facilmente omitida sem prejudicar o algoritmo.

(B)

```
17.       se (posicao = "Janela")
18.          achou ← FALSO
19.          para i ← 0 até 23 com passo 1
20.              se (janela[i] = 0)
21.                  imprima_pl(i)
22.                  achou ← VERDADEIRO
23.              fim_se
24.          fim_para
25.       fim_se
```

Esta parte do código é referente à escolha do usuário, ele entra com algum determinado valor entre 1 e 24, e o programa executa uma busca entre as componentes do vetor `janela[i]`, para saber se esse lugar está disponível.

(C)

```
29.       se (posicao = "Corredor")
30.          achou ← FALSO
31.          para i ← 0 até 23 com passo 1
32.              se (corredor[i] = 0)
33.                imprima(i)
34.                achou ← VERDADEIRO
35.              fim_se
36.          fim_para
37.       fim_se
```

Esta parte do código fornece ao usuário os locais que estão disponíveis para a compra, no corredor.

(D) O código utiliza dois vetores, `janela[i]` e `corredor[i]`, não possui estruturas de repetição e não poderia ser escrito de outra forma menos complexa.

(E)

```
09.    para i ← 0 até 23 com passo 1
10.       se (corredor[i] = 0) ou (janela[i] = 0)
11.          achou ← VERDADEIRO
12.       fim_se
13.    fim_para
```

Nesse caso, em especial, operador lógico OU poderia ser substituído por um operador lógico E, de forma que o resultado final não seria afetado.

QUESTÃO 17. Considere a frase “Um sistema é causal ou estável, e não invariante”, a negação da mesma é:

(A) Um sistema não é causal ou não é estável, e é invariante;

(B) Um sistema não é causal e não é estável, e é invariante;

(C) Um sistema é causal não é estável, ou é invariante;

(D) Um sistema não é causal e não é estável, ou é invariante;

(E) Um sistema não é causal e não é estável, ou é invariante.

QUESTÃO 18. Joãozinho estava desenvolvendo um jogo da forca para o seu trabalho de programação. Para não perder tempo ele usou as funções `pegaPalavra()`, `verifica(letra)` e `procurar(letra, palavra)`. A função `pegaPalavra()` retorna uma palavra qualquer de um banco de palavras, a função `verifica(letra)` verifica se a letra passada já foi utilizada durante essa execução do programa e retorna verdadeiro caso contrário, e a função `procurar(letra, palavra)`, dadas uma letra e uma palavra, verifica se essa letra pertence a essa palavra e retorna verdadeiro caso pertença, verifique se o código feito por Joãozinho está correto:

Programa:

```
01. forca()
02.   palavra ← pegaPalavra()
03.   k ← 5
04.   enquanto (k > 0)
05.     se (k = 1)
06.       imprima_pl(“Cuidado! Última Chance”)
07.     fim_se
08.     imprima_pl(“Digite uma letra”)
09.     leia(letra)
10.     se (verifica(letra))
11.       cond ← procurar(letra, palavra)
```

```
12.       se(!cond)
13.         imprima_pl("Você errou, tente outra letra!")
14.         k ← k-1
15.       fim_se
16.     senão se
17.       imprima_pl("Letra já utilizada")
18.     fim_se
19.   fim_enquanto
20.   se(k > 0)
21.     imprima_pl("Parabéns, você acertou a palavra era " + palavra)
22.   senão se (k = 0)
23.     imprima_pl("Você perdeu, acabaram suas tentativas!")
24.   fim_se
25. fim_forca
```

(A) A linha 10 deveria ser substituída por "**se** (!verifica(letra))" para o programa funcionar corretamente;

(B) O código está errado, pois o bloco enquanto pode continuar em loop quando deveria parar;

(C) Se o bloco **se** iniciado na linha 05 fosse colocado após o bloco **senão** iniciado na linha 16, o programa continuaria tendo as mesmas saídas;

(D) O programa de Joãozinho está funcionando corretamente da forma em que está escrito;

(E) Se o "**senão se** (k=0)" na linha 22 fosse substituído por "**senão**", o código de Joãozinho deixaria de tratar algumas exceções e ocasiona erros.

QUESTÃO 19. (RSI - Adaptada) Cicrano estava fazendo dupla em um trabalho da escola com Fulano. Quando Cicrano foi enviar as respostas do trabalho para o email do Fulano, ficou com medo que outras pessoas tivessem acesso e copiassem. Devido a isto, ele fez a seguinte criptografia: para cada LETRA, ele avançou 3 posições. Ou seja, o que era ' a ' virou ' d ', ' b ' virou ' e ', … , ' z ' virou ' c '. Chamamos o número da criptografia de "chave". Nessa questão, em particular, a chave foi 3 (três). Fulano, com preguiça de fazer manual, decidiu fazer um algoritmo que recebe um vetor de **char**, ou seja, em cada posição do vetor 'resposta' o computador guarda o caractere em seu respectivo valor definido na tabela ASCII. Quando o computador vai imprimir, ele converte da tabela ASCII apra caractere e imprime na tela. Assim, veja o algoritmo 'Decriptar' que Fulano fez :

```
Programa:
01. Decriptar(resposta[])
02.       para i ← 0 até (tamanho(resposta) - 1) com passo 1 faça:
03.            resposta[i] ← 'resposta[i]' - 3
04.       fim_para
05.       imprima(resposta[])
06. fim_Decriptar
```

Sabemos que Cicrano enviou toda a resposta correta em uma única linha e que a resposta estava devidamente pontuada. Deste modo, quando Fulano olhou a saída do seu código ele:

(A) Viu que estava tudo certo e copiou a resposta para o trabalho;

(B) Percebeu que a sua função não estava funcionando pois "resposta" não deveria ser um tipo "string";

(C) Percebeu que seu código estava funcionando, porém a saída estava errada. Um dos erros poderia ser devido ao valor da "chave" que estava errado em seu código;

(D) Percebeu que seu código estava funcionando, porém a saída estava errada. Este erro ocorreu devido a linha número 2, já que ele não percorreu a string inteira;

(E) Fulano percebeu que a saída do seu código formava uma frase desconexa.

QUESTÃO 20. Algoritmos de ordenação são uma parte importante da computação e têm como objetivo a ordenação de valores armazenados, geralmente em vetores. Temos dois exemplos abaixo, onde A[ ] é um vetor com os valores a serem ordenados, p refere-se à posição inicial do vetor e q refere-se à posição final do vetor, de tal modo que p < q:

```
Programa:
01.       InsertionSort(A[], p, q)
02.              para i ← p+1 até q com passo 1
03.                     num ← A[ i ]
04.                     j ← i - 1
05.                     enquanto (( j ≥ p) e ( num < A[ j ] ))
06.                           A[ j + 1 ] ← A[ j ]
07.                           j ← j - 1
```

```
08.                    fim_enquanto
09.                    A[ j + 1 ] ← num
10.             fim_para
11.      fim_InsertionSort
```

```
Programa:
01.       BubbleSort(A[], p, q)
02.             para i ← q até p + 2 com passo -1
03.                  para j ← p até i - 1 com passo 1
04.                        se ( A[ j ] > A[ j + 1 ] )
05.                              aux ← A[ j ]
06.                              A[ j ] ← A[ j + 1 ]
07.                              A[ j+1 ] ← aux
08.                        j ← j + 1
09.                  fim_para
11.             fim_para
12.       fim_BubbleSort
```

Em relação à ordenação, qual a principal diferença entre os dois algoritmos acima?

(A) O método para ordenar os valores. No `InsertionSort()`, todo o vetor é analisado para achar o menor valor **x** existente, então **x** troca de posição com o valor que está no primeiro índice do vetor. Depois é feito o mesmo procedimento, só que agora não será analisado todo o vetor, apenas do segundo índice em diante.

Já no `BubbleSort()`, sempre que um valor **x** é analisado, verifica-se os valores anteriores. Quando o valor é maior que **x** ele vai para o índice seguinte, até que, no índice **y**, seja encontrado um valor menor que **x**. Então, **x** é depositado no índice **y+1**;

(B) O `InsertionSort()` organiza os valores em ordem crescente e o `BubbleSort()` em ordem decrescente;

(C) O método para ordenar os valores. No `InsertionSort()`, sempre que um valor **x** é analisado, verifica-se os valores anteriores. Quando o valor é maior que **x** ele vai para o índice seguinte, até que, no índice **y**, seja encontrado um valor menor que **x**. Então, **x** é depositado no índice **y+1**.

Já no `BubbleSort()` vão sendo feitas combinações dois a dois desde o ínicio do vetor, onde o maior vai sendo jogado sempre pra direita, até que finalmente todos os valores estejam ordenados;

(D) O método para ordenar os valores. No `InsertionSort()`, vão sendo feitas combinações dois a dois desde o ínicio do vetor, onde o maior vai sendo

jogado sempre pra direita, até que finalmente todos os valores estejam ordenados.

Já no `BubbleSort()` todo o vetor é analisado para achar o menor valor **x** existente, então **x** troca de posição com o valor que está no primeiro índice do vetor. Depois é feito o mesmo procedimento, só que agora não será analisado todo o vetor, apenas do segundo índice em diante;

(E) A quantidade de vezes que o vetor é percorrido quando já está com os seus valores ordenados.

QUESTÃO 21. (OBI 2005 - Programação nível 1 Adaptada) Helano é um Engenheiro de Computação que decidiu ir coletar e catalogar dados da flora de uma floresta remota. Helano perde-se na floresta com seu computador e um mapa da região separado por pequenos blocos. Para facilitar seu raciocínio, Helano marca no mapa algumas regiões:

- marca a saída com 0 (zero);
- todos os caminhos acessíveis a pé recebem 1 (um);
- todas as regiões perigosas (caminho que João NÃO irá utilizar) recebem o número 2 (dois);
- e o ponto de partida é marcado com 3 (três).

Além disso, Helano decidiu marcar as bordas do mapa como caminho inacessível, para evitar sair da região delimitada pelo seu mapa. Deste modo, teremos uma matriz de N + 1 linhas e M + 1 colunas, sendo N e M números inteiros. Essa matriz será preenchida nas linhas de 1 até N e nas colunas de 1 até M, sendo as linhas 0 e N + 1 e as colunas 0 e M + 1 para as regiões das bordas do mapa. A seguir uma foto ILUSTRATIVA de como uma matriz deverá ser **CORRETAMENTE** preenchida.

2	2	2	2	2	2	2
2	0	1	1	1	1	2
2	2	2	2	2	1	2
2	2	2	1	1	1	2
2	2	1	1	3	1	2
2	2	2	2	2	2	2

Sabe-se que existe pelo menos um caminho que Helano pode utilizar para chegar a saída. E Helano, como um bom engenheiro, decide fazer um programa pra dizer quantos quadrados ele tem que percorrer no MENOR caminho possível. E para fazer isso, precisamos saber o conceito de 'fila'. Basicamente, fila é uma estrutura que nos permite ordenar nossos objetos com a seguinte propriedade: o primeiro elemento a entrar também é o primeiro a sair. Iremos usar algumas funções como: '`cria(fila)`' cria uma fila com o nome 'fila' inicialmente vazia, '`fila.front()`' para mostrar o primeiro elemento da fila, '`fila.desenfileirar()`' para tirar o primeiro elemento da fila e '`fila.enfileirar(x)`' para adicionar o elemento ' x ' na fila. Helano lembrou que já possui um pedaço do código pronto:

Programa:

```
01.   Menor_Caminho(caminho[N+1][M+1],ix,iy)
02.   //caminho[][] é uma matriz de N linhas e M colunas preenchida
como ilustrado acima.
03.   //ix e iy são as coordenadas do início (marcado com 3 (três) na
ilustração).
04.     cria(fila_x)
05.     cria(fila_y)
06.     fila_x.enfileirar(ix) //insere o valor ix na fila_x
07.     fila_y.enfileirar(iy)
08.     vx[] ← {1,-1,0,0}
09.     vy[] ← {0,0,1,-1}
10.     visitado[N+1][M+1]
11.     para i ← 0 até N com passo 1
12.       para j ← 0 até M com passo 1
13.         visitado[i][j] ← 0
14.       fim_para
15.     fim_para
.
.
.
28.           fim_se
29.         fim_se
30.       fim_para
31.     fim_enquanto
32.     imprima_pl(visitado[fila_x.front()][fila_y.front()] - 1)
```

```
33.   fim_Menor_Caminho
```

Facilmente observamos que as linhas 16 à 27 não aparecem no código. Qual dos itens abaixo apresenta o bloco de código que DEVE ser implementado para que o código funcione como esperado? Suponha que a matriz inicializada (caminho) SEMPRE vai ter solução.

(A)

```
16.        visitado[ix][iy] ← 0
17.        enquanto(caminho[fila_y.front()][fila_x.front()] ≠ 0)
18.          x ← fila_x.front()
19.          fila_x.desenfileirar()
20.          y ← fila_y.front()
21.          fila_y.desenfileirar()
22.          para i ← 0 até 3 com passo 1
23.            se(caminho[ y + vy[i] ][ x + vx[i] ] ≠ 2 )
24.              se(visitado[y+vy[i]][x+vx[i]]=0)
25.                fila_x.enfileirar( x + vx[i] )
26.                fila_y.enfileirar( y + vy[i] )
27.                visitado[y+vy[i]][x+vx[i]] ← visitado[y][x] + 1
```

(B)

```
16.        visitado[ix][iy] ← 0
17.        enquanto(caminho[fila_x.front()][fila_y.front()] ≠ 0)
18.          x ← fila_x.front()
19.          fila_x.desenfileirar()
20.          y ← fila_y.front()
21.          fila_y.desenfileirar()
22.          para i ← 3 até 0 com passo 1
23.            se(caminho[ x + vx[i] ][ y + vy[i] ] ≠ 2 )
24.              se(visitado[x + vx[i] ][ y + vy[i] ] = 0)
25.                fila_x.enfileirar( x + vx[i] )
26.                fila_y.enfileirar( y + vy[i] )
27.                visitado[x + vx[i]][y + vy[i]] ← visitado[x][y]
```

(C)

```
16.        visitado[ix][iy] ← 1
17.        enquanto(caminho[fila_y.front()][fila_x.front()] ≠ 0)
18.          x ← fila_x.front()
19.          fila_x.desenfileirar()
20.          y ← fila_y.front()
```

```
21.          fila_y.desenfileirar()
22.          para i ← 3 até 0 com passo 1
23.          se(caminho[ y + vy[i] ][ x + vx[i] ] ≠ 2 )
24.            se(visitado[y + vy[i] ][ x + vx[i] ] = 0)
25.              fila_x.enfileirar( x + vx[i] )
26.              fila_y.enfileirar( y + vy[i] )
27.              visitado[y + vy[i]][x + vx[i]] ← visitado[y][x] + 1
```

(D)

```
16.        visitado[ix][iy] ← 1
17.        enquanto(caminho[fila_x.front()][fila_y.front()] ≠ 0)
18.          x ← fila_x.front()
19.          fila_x.desenfileirar()
20.          y ← fila_y.front()
21.          fila_y.desenfileirar()
22.          para i ← 0 até 3 com passo 1
23.            se(caminho[ x + vx[i] ][ y + vy[i] ] ≠ 2 )
24.              se(visitado[x + vx[i] ][ y + vy[i] ] = 0)
25.                fila_x.enfileirar( x + vx[i] )
26.                fila_y.enfileirar( y + vy[i] )
27.                visitado[x+vx[i]][y + vy[i]] ← visitado[x][y] + 1
```

(E)

```
16.        visitado[iy][ix] ← 1
17.        enquanto(caminho[fila_x.front()][fila_y.front()] ≠ 0)
18.          x ← fila_x.front()
19.          fila_x.desenfileirar()
20.          y ← fila_y.front()
21.          fila_y.desenfileirar()
22.          para i ← 0 até 3 com passo 1
23.            se(caminho[ x + vx[i] ][ y + vy[i] ] ≠ 2 )
24.              se(visitado[x+vx[i]][y+vy[i]] = 0)
25.                fila_y.enfileirar( x + vx[i] )
26.                fila_x.enfileirar( y + vy[i] )
27.                  visitado[x + vx[i]][y + vy[i]] ← visitado[x][y] +
1
```

QUESTÃO 22. O algoritmo `QuickSort()` é um famoso algoritmo de ordenação de números, que recebe como parâmetros:

- `vetor[]`, que representa um vetor de números;
- `inicio`, que representa o índice inicial do vetor;

- `fim`, que representa o índice final do vetor,

No algoritmo, a variável `posicao_meio` recebe a parte inteira da operação. Exemplo: `inicio = 2` e `fim = 5`, `posicao_meio` receberá 3.
No final do algoritmo, ele retorna o vetor em ordem crescente.
Nesta implementação o vetor começa do índice zero.

Programa:

```
01.     Quick_Sort(vetor[], inicio, fim)
02.         aux ← 0
03.         i ← inicio
04.         j ← fim
05.         posicao_meio ← (início + fim) / 2
06.         meio ← vetor[posicao_meio]
07.         enquanto(i < j)
08.             enquanto(vetor[i] ≤ meio)
09.                 i++
10.             fim_enquanto
11.             enquanto(vetor[j] > meio)
12.                 j--
13.             fim_enquanto
14.             se(i < j)
15.                 aux ← vetor[i]
16.                 vetor[i] ← vetor[j]
17.                 vetor[j] ← aux
18.             fim_se
19.         i++
20.         j--
21.         fim_enquanto
22.         se(inicio < j)
23.             Quick_Sort(vetor, inicio, j)
24.         fim_se
25.         se(i < fim)
26.             Quick_Sort(vetor, i, fim)
27.         fim_se
28.     fim_Quick_Sort
```

Das afirmações sobre o Algoritmo Quick_Sort, assinale a verdadeira.

(A) Esse algoritmo não funciona caso haja números repetidos no vetor;

(B) Não há diferença no tempo de execução entre ordenar um vetor que esteja em ordem decrescente e um vetor que só tem 2 números trocados de lugar;

(C) Caso a variável aux não fosse inicializada com 0, o algoritmo não

conseguiria organizar o vetor;

(D) O algoritmo utiliza o método dividir e conquistar, que consiste em dividir um problema em problemas menores e resolvê-los separadamente;

(E) Um dos motivos da eficiência do algoritmo Quick_Sort é o fato de não haver recursividade em seu código.

QUESTÃO 23. Durante uma aula de matemática, o professor de Lucas falou sobre números palíndromos, números que são iguais tanto sendo lidos da esquerda para a direita como lidos da direita para a esquerda, 4554 e 373 são exemplos de números palíndromos. Lucas é um aluno curioso e resolveu pesquisar mais sobre estes números e criou um algoritmo para verificar se um número é palíndromo ou não. Ao executar o código, Lucas viu que este não estava executando corretamente, o que Lucas deve fazer para corrigir o código?

Programa:

```
01. Palindromo(numero)
02.   palindromo ← numero
03.   x ← tamanho(numero)
04.   se (x = 1)
05.     retorne VERDADEIRO
06.   senão
07.      para i ← 0 até x - 1 com passo 1
08.           aux ← palindromo[x-1-i]
09.           palindromo[x-1-i] ← palindromo[i]
10.           palindromo[i] ← aux
11.     fim_para
12.   fim_se
13.   se (palindromo = numero)
14.          retorne VERDADEIRO
15.   fim_se
16. fim_Palindromo
17.
18. Principal
19.   leia(numero)
20.   se (Palindromo(numero))
21.     imprima_pl("O número é um palíndromo!")
22.   fim_se
23. fim_Principal
```

(A) Não há nenhum erro no código;

(B) Na linha 07, o correto seria: `para i ← 0 até x/2 - 1 com passo 1`;

(C) Nas linhas 08 e 09, o correto seria `palindromo[x-i]` ao invés de `palindromo[x-1-i]`;

(D) Nas linhas 08 e 09, o correto seria `palindromo[x-1]` ao invés de `palindromo[x-1-i]`;

(E) Na linha 07, o correto seria: `para i ← 0 até x/2 com passo 1`.

QUESTÃO 24. Um pergaminho ancião com o chamado *"Algoritmo Lendário da Inutilidade".* Esse algoritmo foi desenvolvido com a única intenção de confundir o leitor. As lendas dizem que esse algoritmo foi utilizado há muitos anos atrás para resolver absolutamente nenhum problema, mas ele funciona perfeitamente.

Programa:

```
01.  AlgorLendInutil(constante)
02.        se (constante % 2 = 0)
03.              constante2 ← constante * 2
04.        senão
05.              constante2 ← constante * 4
06.        fim_se
07.        vetor[constante2]
08.        para i ← 0 até tamanho(vetor)-1 com passo 2
09.              vetor[1] ← i * i
10.              constante ← constante - i
11.              variavel ← 0
12.              enquanto (constante2 > constante)
13.                    constante2 ← constante2 - (i + 1)
14.                    variavel ← variavel + constante2
15.              fim_enquanto
16.              constante2 ← constante * 2
17.        fim_para
18.        imprima(variavel + vetor[1])
19.  fim_AlgorLendInutil
```

Se você, por algum motivo que desconhecemos, desejasse executar este algoritmo usando os valores **3**, **5 e 6** como parâmetro, qual seriam as saídas?

(A) 225, 324, 625;

(B) 225, 625, 100;

(C) 100, 324, 100;

(D) 100, 225, 625;

(E) 196, 625, 324.

QUESTÃO 25. Os Algoritmos Recursivos têm um importante papel no mundo da programação. Uma "Função Recursiva" é a denominação dada para uma função que faz uma chamada a si mesma, e pausa sua execução até que essa chamada (e às suas possíveis subsequentes chamadas) termine e retorne algo. Apesar de serem de difícil compreensão, esses algoritmos têm permitido que problemas considerados complicados fossem resolvidos com programas curtos e diretos.

Observe, abaixo, o seguinte algoritmo recursivo para solucionar o problema de encontrar uma rota em um Labirinto no formato de uma matriz. O próprio labirinto é representado pela matriz **`labirinto[][]`** (de dimensão 5x5), onde cada célula desse labirinto é representada pela sua posição **x** (linha) e **y** (coluna). Cada célula pode ter apenas um dos três valores (ou estar vazia): ***'X'***, ***'V'*** ou ***'F'***.

Programa:

```
01.   explorarLabirinto(labirinto[][], x, y)
02.         se (x > 5) ou (x < 0) ou (y > 5) ou (y < 0)
03.                 retorne FALSO
04.         fim_se
05.         se (labirinto[x][y] = 'X') ou (labirinto[x][y] = 'V')
06.                 retorne FALSO
07.         fim_se
08.         se (labirinto[x][y] = 'F')
09.                 retorne VERDADEIRO
10.         fim_se
11.         imprima_pl("(" + x + "," + y + ")")
12.         labirinto[x][y] ← 'V'
13.
14.         se (explorarLabirinto(labirinto, x, y-1) = VERDADEIRO)
15.                 retorne VERDADEIRO
```

```
16.         fim_se
17.         se (explorarLabirinto(labirinto, x, y+1) = VERDADEIRO)
18.                 retorne VERDADEIRO
19.         fim_se
20.         se (explorarLabirinto(labirinto, x-1, y) = VERDADEIRO)
21.                 retorne VERDADEIRO
22.         fim_se
23.         se (explorarLabirinto(labirinto, x+1, y) = VERDADEIRO)
24.                 retorne VERDADEIRO
25.         fim_se
26.         retorne FALSO
27.   fim_explorarLabirinto
```

Agora, assuma que a função acima será executada para o seguinte labirinto abaixo, assuma também que às linhas e colunas começam a partir do 0. A primeira chamada terá **x** e **y** iguais, respectivamente, à linha e coluna da célula marcada com 'S', ou seja, x = 1 e y = 0. Qual seria a saída impressa por essa função?

(A) (1, 0), (1, 1), (0, 1), (1, 1), (1, 2), (2, 2), (3, 2), (3, 3), (3, 4);

(B) (1, 0), (1, 1), (1, 2), (2, 2), (3, 2), (3, 1), (3, 0), (4, 1), (3, 3);

(C) (3, 4), (3, 3), (2, 2), (1, 2), (1, 1), (1, 0);

(D) (1, 0), (1, 1), (1, 2), (2, 2), (3, 2), (3, 3);

(E) O algoritmo entrará em *loop infinito* e irá imprimir repetidamente (1, 0) e (1, 1).