1) Dados os elementos [87, 6, 3, 5, 1, 9, 42, 43, 57, 71, 23,21, 98] simule o procedimento de inserção no Heap Sort de cada um dos valores, considerando o método de ordenação pelo Heap Máximo.

2) Dados os elementos [87, 6, 3, 5, 1, 9, 42, 43, 57, 71, 23, 21, 98] simule o procedimento de inserção no Heap Sort de cada um dos valores, considerando o método de ordenação pelo Heap Mínimo.

3) Considerando a resolução apresentada no Exercício 1, simule a remoção de 5 elementos do Heap, considerando que o processo de retirada de elementos ocorre a partir da raiz.

4) Considerando a resolução apresentada no Exercício 2, simule a remoção de 5 elementos do Heap, considerando que o processo de retirada de elementos ocorre a partir da raiz.

5) Realize a implementação completa do Algorítmo Heap Sort considerando os Exercícios 1 e 2, tanto para o Heap Máximo quanto para o Heap Mínimo.

6) Explique a diferença entre uma árvore binária balanceada e uma árvore binária não balanceada e apresente a diferença com exemplos de uma árvore AVL para uma árvore Binaria.

7) Demonstre o procedimento completo de inserção em uma árvore binária para o seguinte conjunto de dados [80, 52, 81, 63, 78, 79, 80, 65, 33, 32, 46, 78, 99]. Descreva se a árvore é balanceada ou não balanceada.

8) Demonstre cada um dos passos do percurso em pré ordem, em ordem e pós ordem para o Exercício 7.

9) Dado o exercício 7 demonstre o procedimento de remoção de cada um dos seguintes elementos da árvore, respectivamente, 99, 78, 32, 81, 33. Descreva se a árvore é balanceada ou não balanceada.

10) Demonstre cada um dos passos do percurso em pré ordem, em ordem e pós ordem para o Exercício 9.

11) Realize a implementação de uma árvore binária de alocação dinâmica de memória que receba valores inteiros para inclusão e remoção e apresente cada um de seus elementos nos percursos em pré ordem, em ordem e pós ordem.

12) Dado o Exercício 11, implemente o algoritmo Quick Sort, incluindo o método de Particionamento para o percurso pré ordem.

13) Dado o Exercício 11, implemente o algoritmo Merge Sort, incluindo o método de Intercalação para o percurso pós ordem.

14) Realize a implementação do algoritmo de Busca em Largura em um grafo que explore cada um de seus vértices, garantindo que nenhum vértice do grafo será visitado mais do que uma vez.

15) Realize a implementação do algoritmo de Busca em Profundidade em um grafo que explore cada um dos vértices vizinhos a partir da escolha de um vértice arbitrário, também denominado como raiz, dentro da estrutura do grafo criada

16) Resolva cada uma das questões dissertativas e de múltipla escolha abaixo:

Uma empresa trabalha na produção de concreto e terceiriza o serviço de transporte do produto. Os caminhoneiros telefonam para a empresa e registram seu interesse pelo trabalho. Todas as manhãs, os caminhoneiros estacionam o caminhão no pátio da empresa e aguardam sua vez. O atendimento segue o critério de ordem de chegada. Esse processo é, atualmente, controlado pela secretária, que utiliza sua agenda para gerenciar os motoristas diariamente. A empresa, que carrega, no máximo, 10 caminhões por dia, pretende informatizar esse processo.

Para a solução do problema, apresenta-se, a seguir, um pseudocódigo que utiliza o conceito de fila, mantendo os elementos sempre nas primeiras posições do vetor.

```
Algoritmo Fila_Caminhoneiros
início
   var
        caminhoneiros : vetor[1..10] de texto
        total : inteiro
        procedimento inicializa()
        início
           total <- 0
        fim
        função estaVazia() : lógico
        início
           se (total = 0) então
              retorna verdadeiro
           senão
              retorna falso
           fim-se
        fim
        função estaCheia() : lógico
        início
           se (total >= 10) então
              retorna verdadeiro
           senão
              retorna falso
           fim-se
        fim
        procedimento enfileirar(caminhoneiro : texto)
        início
           se (estaCheia() = falso) então
              total <- total + 1
              caminhoneiros[total] <- caminhoneiro
           senão
              imprima("Fila cheia")
           fim-se
        fim
fim
```

Com base nas informações apresentadas, faça o que se pede nos itens a seguir, expondo cada solução em pseudocódigo ou em linguagem de programação.

a) Implemente a função `desenfileirar`, que deve remover e retornar um elemento representado por um caminhoneiro da fila ou a mensagem "Fila vazia" se não houver elementos. (valor: 6,0 pontos)

b) Implemente o procedimento `mostrarFila`, que deve apresentar a lista de elementos, ou seja, os caminhoneiros que estão na fila. (valor: 4,0 pontos)

O fragmento de código C++, a seguir, contém a implementação da inserção de um nó em uma árvore binária.

```
...
struct noArvore {
   int dado;
   noArvore *esquerda;
   noArvore *direita;
};
noArvore *insere(noArvore *arvore, int valor) {
   if (arvore == NULL) {
      arvore = new noArvore;
      arvore->esquerda = NULL;
      arvore->direita = NULL;
      arvore->dado = valor;
   } else if(valor < arvore->dado) {
        arvore->esquerda = insere (arvore->esquerda, valor);
     }
     else {
        arvore->direita = insere (arvore->direita, valor);
     }
     return(arvore);
}
```

Com base no exposto, assinale a opção que apresenta o fragmento de programa para imprimir os valores contidos nos nós, percorrendo a árvore em pré-ordem.

A
```
void preorder(noArvore *raiz) {
   if (raiz != NULL) {
      cout << raiz->dado << " ";
      preorder(raiz->esquerda);
      preorder(raiz->direita);
   }
}
```

B
```
void preorder(noArvore *raiz) {
   if (raiz != NULL) {
      preorder(raiz->direita);
      preorder(raiz->esquerda);
      cout << raiz->dado << " ";
   }
}
```

C
```
void preorder(noArvore *raiz) {
   if (raiz != NULL) {
      preorder(raiz->esquerda);
      cout << raiz->dado << " ";
      preorder(raiz->direita);
   }
}
```

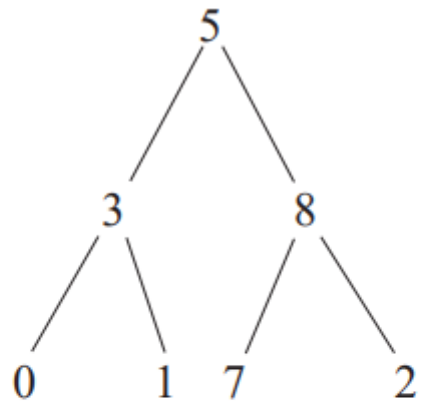
D
```
void preorder(noArvore *raiz) {
   if (raiz != NULL) {
      preorder(raiz->esquerda);
      preorder(raiz->direita);
      cout << raiz->dado << " ";
   }
}
```

E
```
void preorder(noArvore *raiz) {
   if (raiz != NULL) {
      preorder(raiz->direita);
      cout << raiz->dado << " ";
      preorder(raiz->esquerda);
   }
}
```

Existem várias maneiras de se percorrer uma árvore binária. A função a seguir, escrita em pseudo-código, percorre uma árvore na ordem esquerda-raiz-direita, conhecida por varredura e-r-d recursiva. A função `erd()` recebe por parâmetro a raiz `r` de uma árvore, e faz uso de seus elementos `esq`, `dir` e `cont`, que representam, respectivamente, ponteiros para uma sub-árvore à esquerda de `r`, uma sub-árvore à direita de `r` e o conteúdo de `r`, respectivamente.

```
função erd (árvore r)
{
      se ( r != NULO )
      {
            erd( r -> esq );
            escreva ( r-> conteúdo );
            erd( r -> dir );
      }
}
```

Considere a árvore binária a seguir.



A sequência correta de exibição do conteúdo da árvore utilizando a função `erd()` é

A) 5, 3, 8, 0, 1, 7, 2
B) 0, 1, 7, 2, 3, 8, 5
C) 0, 3, 5, 1, 7, 8, 2
D) 0, 3, 1, 5, 7, 8, 2
E) 2, 7, 8, 5, 0, 3, 1

A pilha é uma estrutura de dados que permite a inserção/ remoção de itens dinamicamente seguindo a norma de último a entrar, primeiro a sair. Suponha que para uma estrutura de dados, tipo pilha, são definidos os comandos:

- PUSH (p, n): Empilha um número "n" em uma estrutura de dados do tipo pilha "p";
- POP (p): Desempilha o elemento no topo da pilha.

Considere que, em uma estrutura de dados tipo pilha "p", inicialmente vazia, sejam executados os seguintes comandos:

```
PUSH (p, 10)
PUSH (p, 5)
PUSH (p, 3)
PUSH (p, 40)
POP (p)
PUSH (p, 11)
PUSH (p, 4)
PUSH (p, 7)
POP (p)
POP (p)
```

Após a execução dos comandos, o elemento no topo da pilha "p" e a soma dos elementos armazenados na pilha "p" são, respectivamente,

A) 11 e 29.
B) 11 e 80.
C) 4 e 80.
D) 7 e 29.
E) 7 e 40.

Uma estrutura de dados do tipo pilha pode ser usada em um algoritmo que permite imprimir uma palavra de forma invertida. Exemplo: FELICIDADE deve ser impresso como EDADICILEF.

Utilizando as variáveis declaradas abaixo:

```
pilha[1..50]: caractere;

i, topo: inteiro;

palavra: string;
```

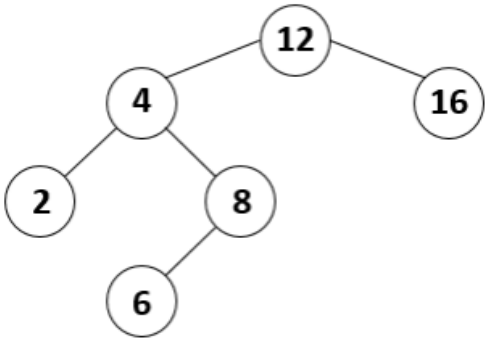
Em pseudocódigo, faça o que se pede nos itens a seguir.

a) Desenvolva a rotina *push* que inclui um elemento na pilha. (valor: 3,0 pontos)

b) Desenvolva a rotina *pop* que retira um elemento da pilha. (valor: 3,0 pontos)

c) Desenvolva a rotina que leia a palavra e, usando a pilha, a imprima de forma invertida. (valor: 4,0 pontos)

QUESTÃO 23 – Considere a árvore binária da figura a seguir:



Os resultados das consultas dos nós dessa árvore binária em pré-ordem e pós-ordem são, respectivamente:

A) (2 4 6 8 12 16) e (2 6 8 4 16 12).
B) (12 4 2 8 6 16) e (2 4 6 8 12 16).
C) (2 6 8 4 16 12) e (12 4 2 8 6 16).
D) (2 4 6 8 12 16) e (12 4 2 8 6 16).
E) (12 4 2 8 6 16) e (2 6 8 4 16 12).

Sobre as estruturas de dados clássicas e seus algoritmos, atribua V (verdadeiro) ou F (falso) às afirmativas a seguir. ( ) A disciplina de acesso da estrutura de dados Pilha determina que o último elemento inserido no conjunto deva ser o primeiro a ser removido. ( ) A implementação de lista utilizando alocação sequencial dos elementos, comparada à alocação encadeada, necessita de mais espaço de armazenamento por elemento do conjunto. ( ) A pesquisa sequencial é mais eficiente que a pesquisa binária para busca de elementos em listas ordenadas implementadas com alocação sequencial dos elementos. ( ) As estruturas de dados Pilha e Fila podem ser implementadas utilizando tanto abordagens baseadas na alocação sequencial quanto na alocação encadeada dos elementos. ( ) A inserção de um elemento no início de uma lista duplamente encadeada implica no deslocamento dos elementos já existentes na memória. Assinale a alternativa que contém, de cima para baixo, a sequência correta.

- **a)** V, V, V, F, F.
- **b)** V, F, V, F, F.
- **c)** V, F, F, V, F.
- **d)** F, V, F, V, V.
- **e)** F, F, V, F, V.