Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Qual será o elemento no topo da pilha após a execução do trecho de código abaixo?

push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

pop(&stack);

pop(&stack);

push(&stack, 40);

push(&stack, 50);

1. 10
2. 20
3. 30
4. 40
5. 50
6. Quais serão os elementos que estão no início e fim da fila respectivamente após a execução do trecho de código abaixo?

enqueue(&queue, 10);

enqueue(&queue, 20);

enqueue(&queue, 30);

dequeue(&queue);

enqueue(&queue, 40);

enqueue(&queue, 50);

dequeue(&queue);

1. 10 e 50
2. 20 e 40
3. 30 e 40
4. 30 e 50
5. 20 e 50
6. Em uma lista simplesmente encadeada, para que serve o trecho de código abaixo?

void func(struct Node\*head, int x) {

struct Node\* newNode = createNode(x);

if (head == NULL) {

head = newNode;

} else {

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

1. Cria um nó na lista em qualquer posição
2. Cria um nó na lista no início da mesma
3. Cria um nó na lista no fim da lista
4. Cria um nó na lista no meio da lista.
5. Cria um nó na lista na posição de x
6. Complete as linhas em branco para inserir um elemento em uma lista duplamente encadeada:

void insertAtEnd(struct DoublyLinkedList\* list, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

perror("Erro na alocação de memória para o nó");

exit(EXIT\_FAILURE);

}

newNode->data = data;

newNode->next = NULL;

if (list->head == NULL) {

newNode->prev = NULL;

list->head = newNode;

list->tail = newNode;

return;

}

newNode->prev = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

list->tail->next = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

list->tail = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

1. O que será impresso no código abaixo:

int f(int n) {

if (n < 0 ) {

return 1;

} else {

return n \* f(n - 2);

}

}

int main() {

int numero = 5;

int resultado = f(numero);

printf("%i", resultado);

}

1. 9
2. 11
3. 13
4. 15
5. 17
6. Qual das seguintes operações não é comumente associada a uma estrutura de pilha (stack)?

A) Inserir elementos no início.

B) Remover elementos do topo.

C) Inserir elementos no final.

D) Verificar se a pilha está vazia.

E) Acessar o elemento no topo da pilha.

1. Em uma fila (queue) implementada com um array, qual operação é usada para adicionar um elemento à fila?

A) Pop

B) Dequeue

C) Push

D) Enqueue

E) Peek

1. Quando você remove um elemento de uma pilha, qual parte da pilha é afetada?

A) Topo

B) Fundo

C) Meio

D) Frente

E) Traseira

1. Considere uma pilha circular implementada com um array de tamanho fixo. A pilha circular tem um ponteiro "top" que aponta para o elemento no topo da pilha. Se "top" estiver no final do array e um novo elemento for inserido, ele será colocado no início do array. Se "top" estiver no início e um elemento for removido, ele será retirado do final do array. Qual é a principal vantagem de usar uma pilha circular em relação a uma pilha convencional com tamanho fixo?

A) Uma pilha circular permite adicionar elementos indefinidamente sem limite de tamanho.

B) Uma pilha circular é mais eficiente em termos de uso de memória.

C) Uma pilha circular evita o problema de estouro de pilha (stack overflow).

D) Uma pilha circular permite acesso aleatório aos elementos.

E) Uma pilha circular não tem vantagens significativas em relação a uma pilha convencional.

1. Você está desenvolvendo um aplicativo de processamento de tarefas de impressão em uma impressora compartilhada. Os usuários enviam tarefas e essas tarefas precisam ser processadas na ordem em que foram recebidas. No entanto, os administradores do sistema também precisam ter a capacidade de cancelar qualquer tarefa que ainda não foi impressa. Qual estrutura de dados você escolheria para implementar a fila de impressão e por quê?

A) Pilha, porque permite adicionar elementos rapidamente no topo e remover elementos do topo, o que é ideal para processamento de tarefas em ordem.

B) Fila, porque segue o princípio "primeiro a entrar, primeiro a sair" (FIFO), que é apropriado para processar tarefas na ordem em que foram recebidas.

C) Lista encadeada, porque permite a inserção e remoção eficiente de elementos em qualquer posição da lista.

D) Lista duplamente encadeada, porque permite a inserção e remoção eficiente de elementos em qualquer posição da lista e também é fácil de percorrer na ordem inversa.

E) Árvore binária, porque permite uma organização hierárquica das tarefas de impressão, o que é útil para priorizar tarefas.

Prova Prática:

Parte 1: Faça um programa que receba 10 números inteiros e armazene os mesmos em uma pilha.

Parte 2: Após receber os 10 valores processe os dados da pilha até que ela fique totalmente vazia. Para cada dado processado da pilha insira o mesmo em uma lista duplamente encadeada.

Parte 3: Assim que a lista estiver preenchida, remova da lista os valores negativos.

Parte 4: No final, mostrar a quantidade de elementos restantes na lista.