Jogo Tetris Stack

Prof. Nathan Alves

Objetivos

- Aplicar um programa em C que implemente a estrutura de dados fila, a fim de simular o gerenciamento da fila de peças do jogo Tetris Stack, controlando inserções, remoções e armazenamento de dados.
- Empregar um programa em C que construa e manipule uma estrutura de dados do tipo pilha, aplicando-a como um sistema de reserva de peças no jogo Tetris Stack que controla a inserção, recuperação e exibição dos dados.
- Aplicar um programa em C que demonstre a comunicação integrada entre fila e pilha, criando um sistema que permita a troca de dados entre as duas estruturas, a reorganização da ordem das peças e a exibição das informações no Tetris Stack.

Introdução

Olá! Boas-vindas! Neste conteúdo, você assumirá o papel de programador(a) responsável pela lógica do jogo Tetris Stack, cujo objetivo é organizar peças em tempo real e de forma estratégica. Para isso, será necessário aplicar conceitos de fila e pilha para controlar a sequência de peças e armazenar temporariamente as que não devem ser jogadas de imediato. Cada estrutura será utilizada com propósito específico, exigindo decisões baseadas na ordem de chegada (fila) e na reserva temporária (pilha).

Nosso estudo será dividido em três etapas, que evoluem de forma gradual em complexidade. Assim, na primeira, você aprenderá a implementar uma fila circular, responsável por organizar as peças prestes a serem jogadas. Na segunda, será a vez de controlar uma pilha de reserva para que o jogador possa guardar e recuperar peças de modo estratégico. Por fim, na terceira, você vai integrar as duas estruturas, criando funcionalidades mais complexas, como a troca de peças ou de toda a pilha com parte da fila, a fim de ter uma reorganização mais dinâmica.

Ao longo do caminho, você irá ver conceitos importantes da linguagem C, como **struct**, **arrays**, **ponteiros**, **modularização** e **controle da memória estática**. Os exemplos propostos seguem uma progressão lógica, sempre contextualizados, de forma que você possa utilizar no jogo. Ao final, você vai compreender como essas estruturas funcionam e por que são utilizadas dessa forma.

O vídeo a seguir, apresenta o cenário do jogo Tetris Stack, contextualizando você dentro da proposta do desafio. Aproveite para ver a missão principal e os três níveis de complexidade.

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

Antes de navegar e avançar pelos níveis do desafio, conheça o cenário onde tudo acontece e descubra qual será a sua missão. Prepare-se!

Cenário

A ByteBros, desenvolvedora de jogos voltados para o ensino de lógica e programação, está desenvolvendo um novo título chamado **Tetris Stack**, um jogo eletrônico no qual os jogadores precisam organizar e posicionar peças utilizando estruturas de dados fundamentais como parte das mecânicas de jogo.

Nele, o jogador assume o papel de um arquiteto digital que precisa montar linhas com peças em tempo real. Entretanto, em vez de apenas reagir, os jogadores também podem reservar, trocar e manipular peças utilizando conceitos de **fila** e **pilha** para definir a ordem e a estratégia de montagem da estrutura.

Sua missão

Designaram você como desenvolvedor(a) técnico(a) responsável pela estrutura de controle de peças do jogo para implementar o sistema de **fila de peças futuras** e a **pilha de peças reservadas**. Seu trabalho é garantir que as regras de entrada, saída e troca entre as estruturas estejam funcionando e reflitam a lógica do jogo.

Para isso, o programa em C deverá gerenciar tais estruturas, utilizando:

- Variáveis
- Structs
- Operadores
- Condicionais
- Estruturas aninhadas
- Funções de entrada e saída

Cada ação do jogador (como jogar, reservar ou recuperar uma peça) deverá ser representada e tratada por sua lógica. Por fim, lembre-se: sua implementação será usada como base para os testes e ajustes futuros no jogo.

Introdução a filas

Acompanhe, no vídeo a seguir, o que são filas, a lógica de funcionamento FIFO e onde elas são aplicadas na computação e no mundo real. Imperdível!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Conceito de filas

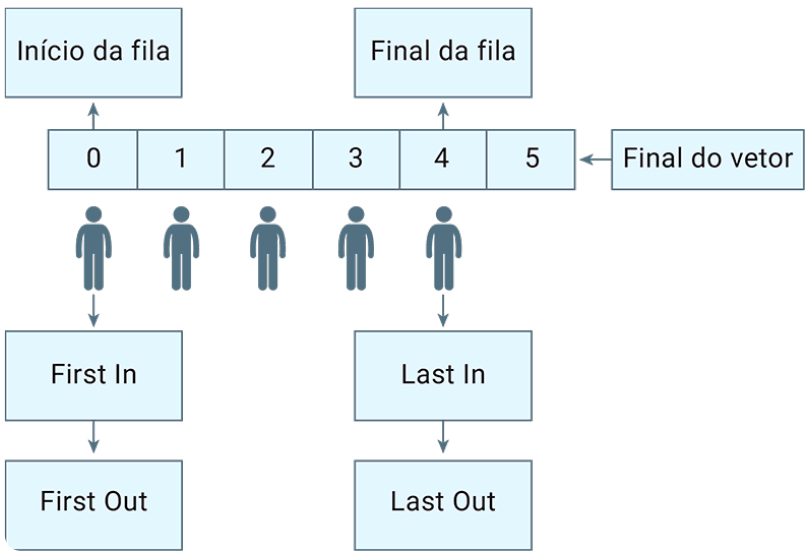
Na programação de computadores, as estruturas de dados são usadas para o desenvolvimento de sistemas eficientes e organizados. Entre elas, podemos destacar a fila por sua atuação em situações que exigem tratamento ordenado de tarefas. Uma vez compreendida sua aplicação, podemos construir ótimas soluções em diversas áreas da computação, como redes, sistemas operacionais e processamento de dados em tempo real.

Vamos entender melhor! A **fila** é uma estrutura de dados que segue o princípio **FIFO (First In, First Out)**. Ou seja, a ideia aqui é que o primeiro elemento a entrar é o primeiro a sair. É por isso que a fila é considerada ideal para o gerenciamento de sequências de tarefas que precisam ser tratadas na ordem em que foram recebidas. Por essa característica, ela é comum em sistemas de atendimento, processamento de requisições, impressoras, transmissão de dados em buffers e comunicação entre processos.

Observe, a seguir, a imagem com uma fila de pessoas, na qual os números acima são seus índices, como se estivessem dentro de um array de pessoas.

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Exemplo de fila.

Agora vamos entender o papel de cada parte da imagem:

Início da fila

Representa o primeiro elemento da fila como o início, mesmo que ele não esteja no índice 0 do array.

Final da fila

Indica o último elemento da fila como o final, ainda que ele não esteja no último índice do array.

First In/First Out

Compreende que o elemento que entra primeiro é o primeiro a sair.

Last In/Last Out

Reconhece que o elemento que entra por último é o último a sair.

As filas modelam o comportamento de várias situações reais. Por exemplo, uma linha em um caixa de supermercado, requisições de clientes em um servidor web ou processos aguardando tempo de CPU. Portanto, lembre-se: saber implementar e manipular uma fila significa ter o controle lógico sobre esse tipo de fluxo ordenado.

Importância

Dominar o uso de filas vai além de sua implementação. Ela é constantemente cobrada em entrevistas técnicas e desafios de lógica, pois representa uma estrutura básica do pensamento computacional. Além disso, a fila está presente em algoritmos mais complexos, como busca em largura (BFS) em grafos, sistemas de filas em simulações e gerenciamento de prioridades em sistemas operacionais.

> Muitas linguagens modernas forneçam implementações prontas. Mas compreender os detalhes de funcionamento (por exemplo, como detectar uma fila cheia ou vazia) e gerenciar os ponteiros de início e fim permitem que você faça escolhas mais eficientes em relação a performance e uso de memória.

As filas são indispensáveis para a organização lógica de processos que devem ser tratados em ordem cronológica. Seja em um jogo com eventos em tempo real, fila de requisições em rede ou gerenciamento de tarefas de um robô, sua aplicação traz fluidez, organização e previsibilidade. O domínio dessa estrutura habilita o programador a construir sistemas mais coesos e estruturados, além de promover uma mentalidade lógica e ordenada na resolução de problemas computacionais.

Lógica circular

Confira o vídeo a seguir, que demonstra, de forma prática, como funciona a lógica circular e o cálculo que ela representa. Veja a importância dessa técnica para manter o uso cíclico de um array de tamanho fixo para o uso de filas.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O que é lógica circular?

Quando usamos um array como base de uma fila, surgem limitações naturais. Por exemplo, ao realizar múltiplas inserções e remoções, o índice **fim** pode alcançar o final do array, mesmo que ainda existam posições no início liberadas por operações de remoção (dequeue).

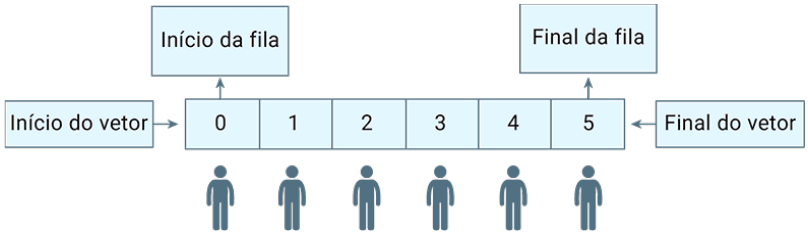
Para resolver isso, aplicamos a **lógica circular** (array circular), ou seja, o array é tratado como se estivesse conectado em seus extremos. Assim, ao ultrapassarmos o último índice, voltamos ao início do array.

Lógica circular na prática

Vamos utilizar uma fila de pessoas para visualizarmos na prática o comportamento de uma fila circular. A imagem a seguir representa uma fila com capacidade para 6 elementos (pessoas). Após inserir todos eles, o **fim** fila atinge o índice 5. Veja!

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.

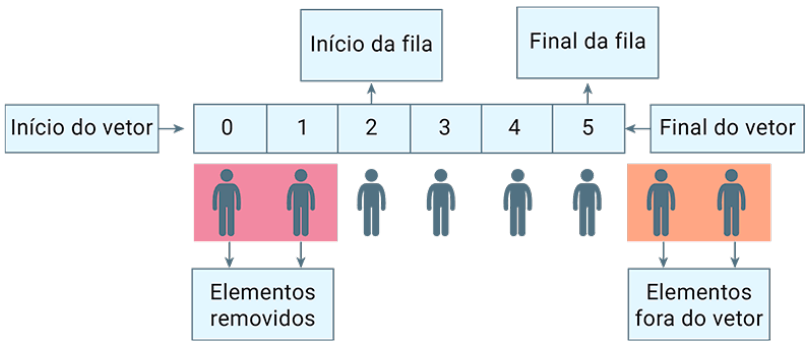


Fila circular 1.

Se removermos dois itens, o **início** se move para a posição 2. Porém, ao tentarmos inserir mais elementos, o índice **fim** não pode avançar para o 6 ou 7, pois essas posições não existem no array, como podemos observar adiante.

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Fila circular 2.

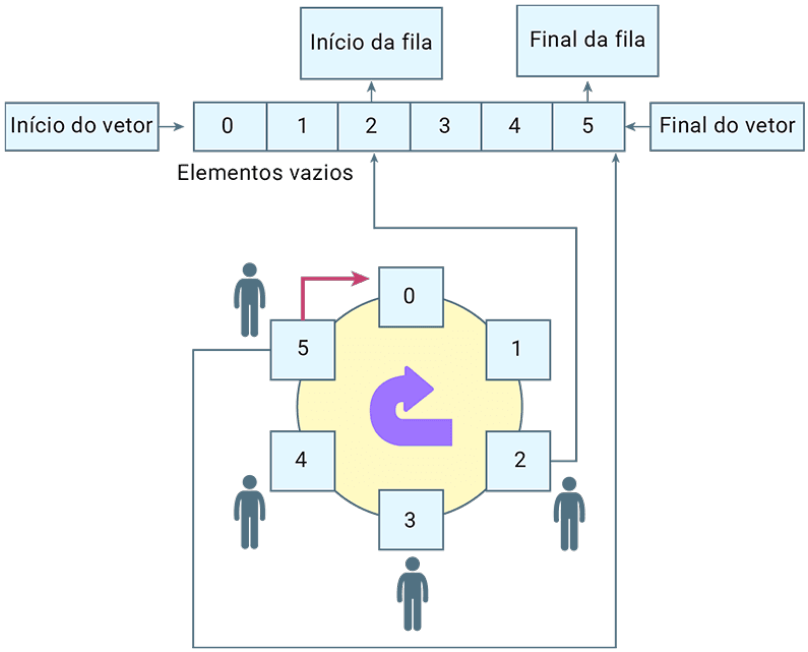
É nesse momento que utilizamos o operador módulo %:

$$f->fim = (f->fim + 1)\%$$ MAX;

Com esse cálculo, se somarmos 1 ao índice **fim** (atualizar o final da fila), ele retornará a 0 quando ultrapassar o tamanho máximo (**MAX**), reiniciando o ciclo do array. Essa operação torna a visualização do array da seguinte forma:

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Fila circular 3.

Conseguimos reutilizar o espaço já liberado do início do array, mesmo após muitas inserções e remoções. O mesmo conceito é aplicado no valor **início** durante uma remoção. Confira!

$$f->inicio = (f->inicio + 1)\%$$ MAX

Por que isso é importante?

Para que filas implementadas com arrays fixos não desperdicem memória! Sem o uso cíclico, precisaríamos realocar manualmente os elementos após cada remoção para “empurrá-los” para o início, o que seria ineficiente.

Portanto, o uso cíclico evita esse custo e mantém as operações com complexidade constante; veja:

Inserção (enqueue): $O(1)$ Remoção (dequeue): $O(1)$

Operações essenciais em filas

Veja, no vídeo a seguir e de forma prática, como funcionam as operações básicas em uma fila: inserir (enqueue), remover (dequeue) e espiar (peek). Confira também como cada função atua sobre a estrutura e quais cuidados devem ser tomados para evitar erros.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução às operações básicas

Filas são estruturas de dados que funcionam segundo o princípio FIFO, ou seja, o primeiro elemento inserido é o primeiro a ser retirado. Portanto, isso promove um comportamento ordenado e previsível, certo?

Para dominarmos o uso de filas, precisamos conhecer bem as operações básicas que controlam seu funcionamento: inserir, retirar e espiar. A seguir, detalharemos cada uma delas, abordando sua finalidade, seu comportamento e uso correto. Vamos lá!

Inserir (enqueue)

Significa colocar um novo elemento no final da fila. Porém, essa função precisa verificar se ainda há espaço disponível antes de prosseguir. Em implementações com arrays, isso é feito por meio de controle do total de elementos ou de um índice de fim, como podemos observar a seguir.

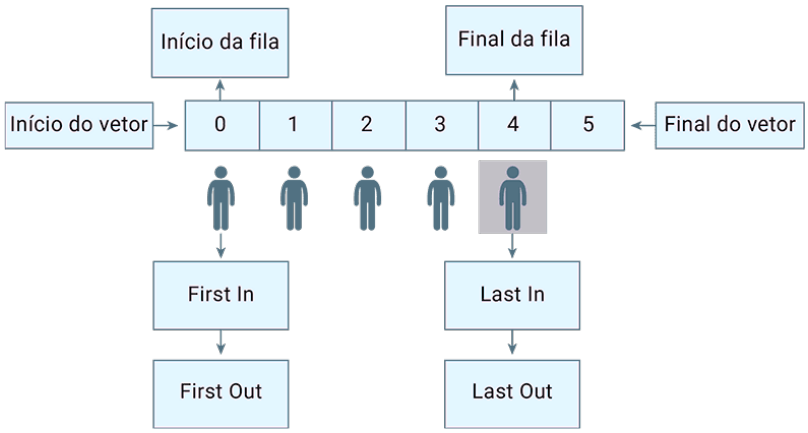
```
c

void inserir(Fila *f, Elemento e) {
    if (filaCheia(f)) // Impede inserção se estiver cheia
        return;
    f->itens[f->fim] = e;            // Insere no final
    f->fim = (f->fim + 1) % MAX;     // Atualiza circularmente o índice
    f->total++;                      // Incrementa a contagem de elementos
}
```

Inserir em array é uma operação fundamental para alimentar a fila com dados. O uso do operador módulo (%) garante um comportamento circular, evitando desperdício de espaço. Veja!

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Fila enqueue.

Na imagem, podemos analisar o funcionamento da função **inserir**: um novo elemento é adicionado no final da fila, e o índice que representa o final dela é modificado.

Resumindo

Inserir adiciona um novo item no final da fila, respeitando o limite máximo e mantendo a ordem dos dados.

Retirar (dequeue)

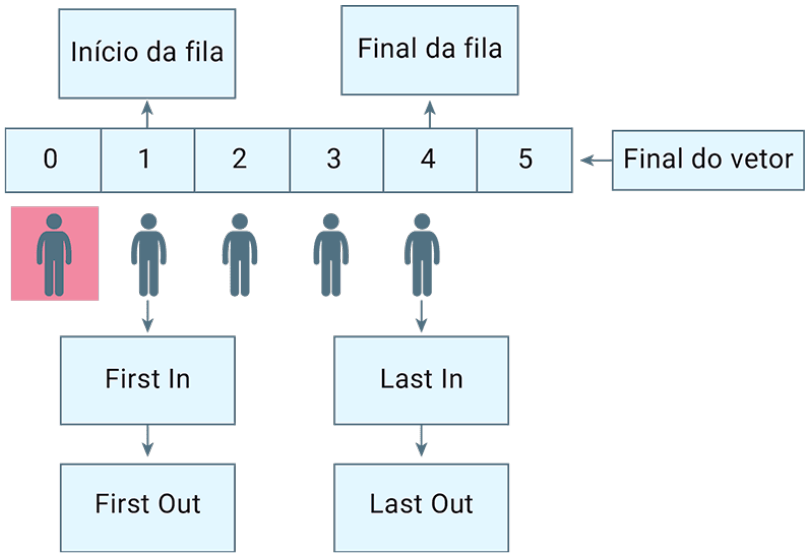
Significa eliminar o elemento mais antigo, localizado na frente. Mas atenção: antes de remover, é necessário verificar se a fila não está vazia, evitando acessar posições inválidas. Acompanhe!

```c
Elemento retirar(Fila *f) {
    Elemento vazio = {-1};              // Valor padrão caso a fila esteja vazia
    if (filaVazia(f))
        return vazio;      // Evita remoção se estiver vazia
    Elemento e = f->itens[f->inicio];    // Armazena o item a ser removido
    f->inicio = (f->inicio + 1) % MAX;   // Atualiza o índice de início
    f->total--;                          // Diminui o total
    return e;                            // Retorna o item removido
}
```

Quando queremos processar elementos por ordem de chegada, precisamos recorrer à função **retirar**. O controle circular do índice garante eficiência mesmo com uso repetido. Acompanhe!

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Fila dequeue.

Na imagem, podemos analisar o funcionamento da função **retirar**: o início da fila é removido (pois era o primeiro a sair), atualizando o novo índice, que representará o novo início da fila.

Resumindo

Retirar exclui o primeiro item da fila e atualiza os índices de forma segura.

Verificar sem remoção (peek)

Em certas situações, é importante olhar o que está no início da fila sem removê-lo. Isso é feito pela operação de "espiar", também conhecida como *peek*. A função apenas retorna o elemento da frente sem alterar nenhum índice, como podemos ver no código a seguir.

```c
Elemento espiar(Fila *f) {
    Elemento vazio = {-1};              // Valor padrão se a fila estiver vazia
    if (filaVazia(f))
        return vazio;     // Impede acesso indevido
    return f->itens[f->inicio];         // Retorna o elemento da frente
}
```

A função é útil para tomar decisões baseadas no próximo item, mas sem comprometer a fila, como verificar quem é o próximo da vez ou validar um valor.

Resumindo

Espiar acessa o primeiro item da fila sem removê-lo, sendo ideal para inspeções seguras.

Inserir, retirar e espiar compõem o núcleo funcional de uma fila. Com elas, é possível controlar com precisão a entrada e saída de elementos em diversos contextos computacionais. A partir da implementação correta, a fila se comporta de forma estável, evitando erros de acesso e perdas de dados.

Atenção

Todas as funções precisam trabalhar de forma integrada, realizando verificações de estado — como identificar se a fila está cheia ou vazia — para que o código funcione como pretendido. Uma fila bem manipulada evita falhas e torna os processos mais previsíveis, organizados e eficientes.

Dominar o uso dessas funções ajuda na criação de sistemas mais confiáveis, inclusive em situações que exigem ordem e controle de fluxo, como agendamentos, gerenciamento de eventos, filas de processamento e muito mais.

Declaração, implementação e uso da fila

O vídeo a seguir ensina como declarar e implementar uma fila na linguagem C e inicializar, inserir e remover elementos, além de exibir a estrutura no console com clareza. Não perca!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Fundamentos da construção e do uso das filas

Vamos, a partir de agora, compreender como construir, utilizar e manipular uma fila em linguagem C de forma prática e técnica. Começaremos pela criação das estruturas necessárias, incluindo a definição dos elementos que serão armazenados e da própria estrutura da fila. Em seguida, veremos como inicializar essa estrutura na memória, para que os valores de controle estejam preparados para o uso.

Avançaremos para a implementação das funcionalidades fundamentais de uma fila, como inserir elementos, remover elementos e exibir seu conteúdo de forma organizada no console. Também discutiremos como identificar os estados de fila cheia e fila vazia para evitar falhas na execução do programa. Por fim, incluiremos uma função para geração automatizada de dados, útil em simulações e testes, e mostraremos como tudo isso se integra ao **main**.

Criando uma fila

Neste exemplo, vamos implementar uma fila de pessoas, na qual cada uma terá um nome (char) e idade (int). Além disso, vamos ver também parte de um código, e o conjunto resultará na implementação completa de uma fila, etapa por etapa. Dica: atente-se à estrutura e ao que ela precisa para funcionar, ok?

Struct do elemento

O primeiro passo para utilizar uma fila com mais significado é definir o tipo de dado que será armazenado. No nosso exemplo, cada item da fila representa uma pessoa.

```
c

typedef struct {
    char nome[30];
    int idade;
} Pessoa;
```

Lembre-se: esse tipo estruturado facilita a organização dos dados e resulta no uso de informações mais completas nos elementos da fila.

Struct da fila

A fila em si precisa de um array para armazenar os elementos e as variáveis de controle para saber onde inserir e remover. Acompanhe!

```
c

#include
#include

#define MAX 5

typedef struct {
    Pessoa itens[MAX];
    int inicio;
    int fim;
    int total;
} Fila;
```

A struct Fila guarda os dados e controla o estado da fila com índices e contador de elementos.

Inicializar fila

Antes de usar a fila, seus índices e contador precisam ser configurados, como podemos ver a seguir.

```
c

void inicializarFila(Fila *f) {
    f->inicio = 0;
    f->fim = 0;
    f->total = 0;
}
```

A inicialização define o ponto de partida da fila, visando ao seu funcionamento correto desde o início.

Verificar fila cheia e/ou vazia

É importante para sabermos se a fila atingiu a capacidade máxima. Confira a seguir!

```c
int filaCheia(Fila *f) {
    return f->total == MAX;
}
```

Fazer tal verificação também confirma se há elementos disponíveis para remoção. Veja!

```c
int filaVazia(Fila *f) {
    return f->total == 0;
}
```

Fila cheia impede novas inserções. Logo, essa verificação evita sobreposição de dados. Fila vazia, por sua vez, impede remoções, o que é importante para evitar acesso a posições inválidas.

Inserir (enqueue) e remover (dequeue)

Enqueue e **dequeue** são as funções que trazem a característica FIFO para a estrutura de dados fila. A primeira adiciona novo item no final da fila, enquanto a outra remove o item que representa o início da fila. Portanto, inserções e remoções não seriam possíveis sem essas duas implementações.

Enqueue adiciona um novo item ao final da fila, se houver espaço. Observe o código adiante!

```c
void inserir(Fila *f, Pessoa p) {
    if (f->total == MAX) {
        printf("Fila cheia. Não é possível inserir.\n");
        return;
    }

    f->itens[f->fim] = p;
    f->fim = (f->fim + 1) % MAX;
    f->total++;
}
```

Confira agora como dequeue remove o elemento da frente da fila:

```c
void remover(Fila *f, Pessoa *p) {
    if (filaVazia(f)) {
        printf("Fila vazia. Não é possível remover.\n");
        return;
    }

    *p = f->itens[f->inicio];
    f->inicio = (f->inicio + 1) % MAX;
    f->total--;
}
```

Dequeue insere no final e atualiza os controles da fila, e o uso do módulo mantém a circularidade. A remoção retorna o primeiro elemento e ajusta a posição de início da fila.

Mostrar fila

Exibe todos os elementos na ordem em que foram inseridos, como podemos ver adiante.

```c
void mostrarFila(Fila *f) {
    printf("Fila: ");
    for (int i = 0, idx = f->inicio; i < f->total; i++, idx = (idx + 1) % MAX) {
        printf("[%s, %d] ", f->itens[idx].nome, f->itens[idx].idade);
    }
    printf("\n");
}
```

Essa função percorre a fila desde o início até o total visto atualmente, respeitando a ordem dos dados.

Declaração e exibição no main()

A fila pode ser usada no main após ser inicializada, inserindo pessoas e mostrando o conteúdo. Acompanhe!

```c
int main() {
    Fila f;
    inicializarFila(&f);  // Inicializa a fila

    // Insere algumas pessoas na fila
    Pessoa p1 = {"João", 25};
    Pessoa p2 = {"Maria", 30};
    inserir(&f, p1);
    inserir(&f, p2);

    mostrarFila(&f);  // Mostra a fila antes da remoção

    // Remove uma pessoa da fila
    Pessoa removida;
    remover(&f, &removida);  // Aqui usamos a função de remoção

    printf("Pessoa removida: %s, %d\n", removida.nome, removida.idade);

    mostrarFila(&f);  // Mostra a fila após a remoção

    return 0;
}
```

Por fim, a sequência que declara, inicializa, insere e exibe mostra o uso prático da fila com clareza.

Atenção

Compreender como declarar, inicializar e manipular filas é necessário para manter a organização dos dados sequencial e controlada, sendo os conceitos de fila cheia, vazia, inserção e remoção a base de seu funcionamento. Mostrar a fila no console facilita o entendimento visual e a depuração de erros. Utilizar estruturas bem definidas e funções organizadas favorece o reuso e a escalabilidade da estrutura, seja em projetos simples ou complexos.

Hora de codar

No vídeo a seguir, veja a implementação de uma fila circular em C de pessoas. Essa prática é fundamental para criar a fila das peças no Tetris Stack, necessitando apenas da criação de uma estrutura que represente peças.

Assim, aproveite para conferir a aplicação prática dos conceitos estudados, que também serve como revisão e reforço da implementação com testes no terminal.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nivel novato

O vídeo a seguir lança o primeiro desafio: criar um programa em C que simula uma fila de peças do Tetris Stack. Com esse conhecimento, você vai implementar inserções e remoções automáticas de peças com identificação e tipo, observando o comportamento FIFO.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O que você vai fazer

A ByteBros, desenvolvedora de jogos voltados para o ensino de lógica e programação, está desenvolvendo um novo título chamado **Tetris Stack**, um jogo eletrônico no qual os jogadores precisam organizar e posicionar peças utilizando estruturas de dados fundamentais como parte das mecânicas de jogo.

Designaram você como desenvolvedor(a) técnico(a) responsável pela estrutura de controle de peças do jogo, incluindo a implementação do sistema de **fila de peças futuras**. Seu trabalho é verificar se as regras de entrada, saída e troca entre as estruturas estão funcionando e se refletem a lógica do jogo.

Agora, você implementará um programa em C que **simula a fila de peças futuras** do Tetris Stack. As peças têm um tipo (como 'I', 'O', 'T', 'L'), que representa suas formas e um identificador numérico (id) exclusivo. O

programa oferecerá ao jogador a opção de **visualizar a fila**, **jogar uma peça (remoção da frente)** ou **adicionar uma nova peça ao final** da fila.

Requisitos funcionais

Seu programa em C deverá:

- **Inicializar a fila de peças** com um número fixo de elementos (por exemplo, 5).

Permitir as seguintes ações:

- **Jogar uma peça,** isto é, remove a peça da frente da fila (dequeue).
- **Inserir nova peça** ao final da fila (enqueue), se houver espaço.
- **Exibir o estado atual da fila** após cada ação, mostrando o tipo (nome) e o id de cada peça.

Atributos das peças

Cada uma possui:

- **nome**: caractere que representa o tipo da peça ('I', 'O', 'T', 'L').
- **id**: número inteiro único que representa a ordem de criação da peça.

Lembre-se: as peças são geradas automaticamente por uma função chamada **gerarPeca**.

Exemplo de saída

Confira a seguir seu estado:

Fila de peças
[T 0] [O 1] [L 2] [I 3] [I 4]

Tabela: Visualização a fila de peças.
Curadoria de TI.

Opções de ação:

Código	Ação
1	Jogar peça (dequeue)
2	Inserir nova peça (enqueue)
0	Sair

Tabela: Comandos disponíveis para manipular a fila de peças.
Curadoria de TI.

Requisitos não funcionais

Observe os seguintes elementos importantes:

- **Usabilidade:** a saída do programa deve ser clara e fácil de entender.
- **Legibilidade:** o código deve ser bem organizado, com comentários explicando a lógica utilizada. Para isso, utilize nomes descritivos de variáveis.
- **Documentação:** o código deve ser comentado, a fim de explicar o propósito de cada parte.

Simplificações para o nível básico

Veja a seguir as principais considerações:

- O foco é somente na estrutura de fila.
- A pilha de peças reservadas não será implementada neste desafio.
- O menu é simples, com três opções fixas: jogar peça, inserir nova peça e sair.
- As peças são geradas de forma automática, e não inseridas manualmente pelo jogador.

Conceitos trabalhados

Os pontos fundamentais são:

- **Fila circular:** manipulação eficiente de elementos com reaproveitamento de espaço.
- **Structs e arrays:** definição e uso de tipos personalizados para representar peças.
- **Entrada e saída de dados:** interação com o jogador via terminal.
- **Funções e modularização:** separação de responsabilidades no código.
- **Operadores lógicos e condicionais:** controle de fluxo para validação de operações e restrições.

Entregando seu projeto

1. **Desenvolva seu projeto no GitHub:** use sempre o mesmo repositório do GitHub.
2. **Atualize o arquivo do seu código:** atualize o arquivo super_trunfo.c com o código completo, incluindo as novas funcionalidades.
3. **Compile e teste seu programa:** faça isso com rigor, garantindo que todas as comparações e cálculos estejam corretos.
4. **Faça commit e push:** faça commit das suas alterações e envie (push) para o seu repositório no GitHub.
5. **Envie o link do repositório no GitHub:** faça isso por meio da plataforma SAVA.

Tutorial git

Você está prestes a aplicar os conceitos aprendidos para resolver um desafio prático no ambiente do GitHub. Veja, a seguir, as instruções gerais para acessar, aceitar e executar o desafio, de modo que sua solução esteja bem estruturada e documentada.

Dê o primeiro passo

Acesse o GitHub Classroom. Nesse ambiente, você terá acesso ao repositório padrão do desafio. Caso ainda não tenha uma conta no GitHub, não se preocupe: você pode criar uma gratuitamente, clicando no link.

Aceite o desafio

Tenha, em seguir, acesso ao repositório no GitHub, no qual você encontrará o repositório criado para o desenvolvimento do seu desafio.

Acesse o repositório

Clique no link do repositório para abrir o ambiente GitHub com a descrição do desafio e a estrutura do modelo de arquivos e pastas que deve ser utilizada. Lembre-se: é esse o link que você deve enviar no SAVA.

Explore a estrutura do ambiente

Veja a estrutura organizada de pastas e arquivos necessários para o desenvolvimento do desafio.

Desenvolva o desafio

Utilize o GitHub CodeSpace para editar o arquivo do código-fonte e desenvolver o desafio. Certifique-se de que o código esteja organizado e funcional para resolver o problema proposto.

Entregue o desafio

Forneça o repositório do GitHub com todos os arquivos de código-fonte e conteúdos relacionados ao projeto. Certifique-se de que o repositório esteja bem estruturado, com pastas e arquivos nomeados de maneira clara e coerente. Envie o link para o repositório do seu desafio no GitHub.

Por último, comente todos os arquivos de código-fonte, pois isso ajuda a demonstrar o quanto você sabe sobre o funcionamento do código e facilita a correção por terceiros. Portanto, seus comentários devem explicar a finalidade das principais seções do código, o funcionamento de algoritmos complexos e o propósito de variáveis e funções utilizadas.

Para finalizar este nível, assista ao vídeo a seguir, que contém dicas e detalhes sobre a entrega do seu projeto.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução a pilhas

O vídeo a seguir introduz o conceito de pilha, destacando seu comportamento LIFO (Last In, First Out) e suas aplicações mais comuns. Ele também prepara o terreno para a implementação prática dessa estrutura na linguagem C. Assista!

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

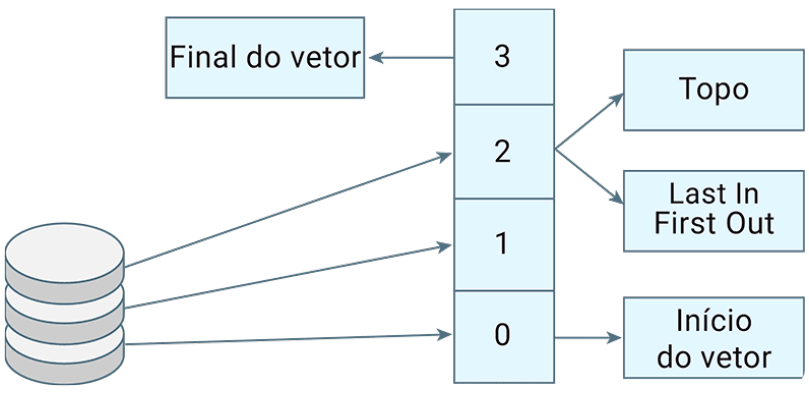
Conhecendo o conceito de pilhas

Quando se trata de manipulação de estados, execução reversa e controle de chamadas, a pilha é uma das estruturas de dados mais conhecidas e interessantes. Na prática da programação, seu uso recorrente em diversas linguagens, algoritmos e arquiteturas faz dela indispensável na formação de qualquer indivíduo programador.

A pilha segue a lógica **LIFO (Last In, First Out)**, ou seja, o último elemento inserido é o primeiro a ser removido. Assim, com essa estrutura, podemos empilhar informações que precisam ser tratadas **em ordem inversa**. Exemplos clássicos incluem a navegação entre páginas da web, a chamada de funções recursivas, o controle de execução de comandos e os sistemas de desfazer/refazer em editores de texto.

Acompanhe a próxima imagem!

> Conteúdo interativo
>
> Acesse a versão digital para ver mais detalhes da imagem abaixo.



Pilha geral.

Na imagem, temos um array de 4 elementos. Ao contrário da fila, que o primeiro elemento adicionado é sempre o primeiro a ser removido, a estrutura de uma pilha reflete um raciocínio natural de empilhar e desempilhar. Assim, cada novo dado é colocado no **topo** e só pode ser removido se estiver nessa posição. Isso exige que o programador pense na lógica de retrocesso de forma organizada e sequencial.

Se imaginarmos uma pilha de pratos (ou uma pilha de peças), sempre colocamos um em cima do outro, e para retirarmos o prato que está na base, teremos que remover toda a pilha antes. Logo, ele será o último a sair, pois foi o primeiro a ser colocado.

A importância do sistema de pilhas

É devido ao fato de serem constantemente utilizadas em algoritmos de:

Parsing

Os algoritmos de parsing analisa e estrutura dados de entrada, verificando sua sintaxe e convertendo os dados para um formato compreensível. O parsing usa uma pilha para organizar tokens em ordem hierárquica, auxiliando na análise sintática de expressões e linguagens.

Resolução de expressões matemáticas

A pilha é fundamental, pois garante a ordem correta dos cálculos. Esse método é utilizado em calculadoras e interpretadores de linguagem.

Backtracking

A pilha é usada para armazenar estados anteriores, possibilitando retornar e tentar alternativas em problemas recursivos. Assim, backtracking é muito utilizado em softwares para que o usuário desfaça ações em sequência.

Os sistemas operacionais utilizam pilhas para o gerenciamento de chamadas de função e armazenamento de variáveis locais. Ao entender como a pilha funciona internamente, podemos otimizar o uso da memória e prever comportamentos críticos, como **estouro de pilha (stack overflow)**.

> Mesmo com suporte nativo em muitas linguagens de programação, compreender a mecânica interna da pilha garante eficiência, segurança e clareza no desenvolvimento de algoritmos que dependem da manipulação de contexto.

Pilhas são mais do que estruturas de armazenamento: elas representam uma **forma de organizar o raciocínio lógico**, ainda mais em situações em que é necessário voltar ou processar dados de forma invertida. Sua aplicação é ampla e varia desde a implementação de **calculadoras** até o **controle de execução de máquinas virtuais**. Dominar seu uso ajuda a projetar algoritmos limpos, eficientes e com grande capacidade de reversibilidade.

Operações essenciais em pilhas

Confira, no vídeo a seguir, as operações básicas das pilhas, como cada uma atua na estrutura e qual seu papel na lógica de reversão de ordem ou controle de contexto.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução ao uso de pilhas

Pilhas são estruturas de dados fundamentais que seguem o princípio LIFO, o que resulta em um comportamento reverso ao das filas. Isso as torna úteis em contextos nos quais queremos desfazer ações, controlar chamadas de funções, converter expressões matemáticas ou navegar entre páginas em um navegador.

Dominar o uso de pilhas requer alto entendimento das operações básicas que controlam seu funcionamento. As principais funções são:

- inserir (push)
- remover (pop)
- consultar o topo (peek)
- liberar (free)

Em um software, temos uma pilha de tamanho fixo armazenando todos as suas ações, para que se houver necessidade, você possa desfazer a última ação feita.

Vamos agora conhecer mais detalhes sobre as principais funções das pilhas!

Inserir (push)

Significa colocar novo elemento no topo. No entanto, a função de inserção precisa verificar se há espaço disponível antes de prosseguir. Em implementações com arrays, o topo é representado por um índice incrementado a cada nova inserção. Veja!

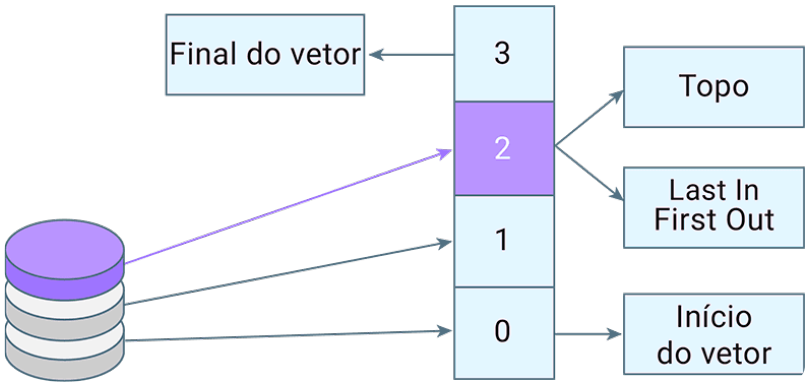
```c
void push(Pilha *p, Elemento e) {
    if (pilhaCheia(p)) {
        printf("Erro: pilha cheia. Não é possível inserir.\n");
        return;
    }

    p->topo++;              // Avança o topo
    p->itens[p->topo] = e;  // Insere o novo elemento
}
```

Observe a imagem a seguir.

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Pilha push.

A operação inserir é usada para construir a pilha. O controle do índice topo assegura que os elementos serão inseridos em ordem e sobrepostos de modo correto.

Resumindo

Push adiciona um novo item no topo da pilha, respeitando o limite máximo e mantendo o controle da ordem inversa. Em softwares, é adicionado um novo elemento a toda ação feita, para que ela possa ser recuperada/desfeita.

Ao digitar a palavra *Pilha*, precisamos de 5 letras, para podermos desfazer a palavra em 5 ações. Assim, precisamos ter uma pilha armazenando cada ação (letra digitada) de forma individual.

Remover (pop)

Significa retirar o que está no topo. A função a seguir recebe um ponteiro no qual o valor removido será armazenado, caso a pilha não esteja vazia. Acompanhe!

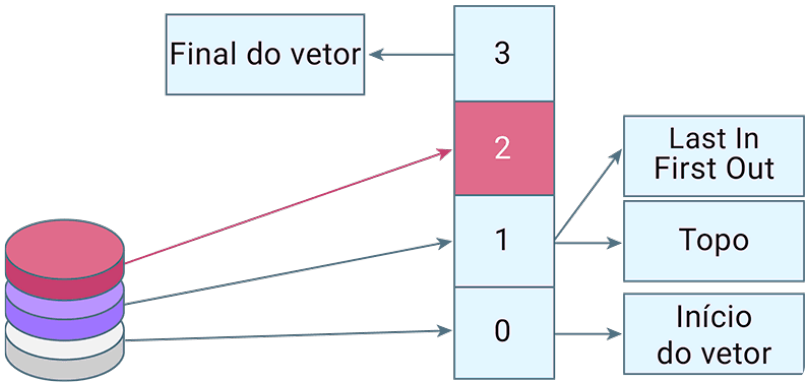
```c
void pop(Pilha *p, Elemento *e) {
    if (pilhaVazia(p)) {
        printf("Erro: pilha vazia. Não há elementos para remover.\n");
        e->valor = -1;  // Define um valor padrão para indicar falha
        return;
    }

    *e = p->itens[p->topo];  // Copia o valor do topo
    p->topo--;               // Decrementa o topo
}
```

Agora, visualize o que acabamos de estudar.

Conteúdo interativo

Acesse a versão digital para ver mais detalhes da imagem abaixo.



Pilha pop.

Vamos entender mais!

Resumindo

Pop remove o item do topo da pilha e o armazena em e, desde que a pilha não esteja vazia. Em softwares, toda vez que o usuário precisa desfazer uma ação, isso é conhecido como removendo o item que está no topo da pilha ao desfazer a última ação feita.

Confira este exemplo: para desfazermos a palavra *Pilha*, precisamos remover o topo da estrutura 5 vezes, e levando em consideração a lógica LIFO, a ordem de remoção seria: a h l i P.

Consulta (peek)

Possibilita consultar o elemento no topo da pilha sem o remover. Isso é útil para inspecionar o último valor inserido. Veja!

```c
void peek(Pilha *p, Elemento *e) {
    if (pilhaVazia(p)) {
        printf("Erro: pilha vazia. Não há elementos no topo.\n");
        e->valor = -1;  // Valor padrão para indicar erro
        return;
    }

    *e = p->itens[p->topo];  // Copia o elemento do topo sem alterar a pilha
}
```

Portanto, peek é uma função que permite acessar informações sem alterar a estrutura.

Resumindo

Peek copia o valor do topo da pilha sem alterar o conteúdo dela, desde que não esteja vazia. Em geral, essa função é utilizada quando há necessidade de exibir o topo da pilha. Na prática, trata-se da ação que será desfeita caso o usuário execute o comando desfazer.

Liberar (free)

Possibilita que você esvazie (ou libere) os dados da pilha inteira. No caso de uma pilha de tamanho fixo, é importante ter em mente que os elementos não são removidos do vetor de modo físico, mas a pilha perde a referência de onde o topo estava. Confira o código a seguir!

```c
// Esvazia a pilha: reinicia o topo
void liberarPilha(Pilha *p) {
    p->topo = -1;  // Define a pilha como vazia novamente
}
```

Em pilhas de tamanho dinâmico, liberar significa utilizar o **free** para desalocar da memória. Veja!

```c
// Libera todos os nós da pilha dinâmica
void liberarPilha(Pilha *p) {
    while (p->topo != NULL) {
        No *remover = p->topo;
        p->topo = p->topo->prox;
        free(remover);  // Libera a memória alocada
    }
}
```

No exemplo, o **while** percorre toda a pilha e depois libera toda a pilha alocada na memória.

Resumindo

A função free() só é necessária em pilhas implementadas com alocação dinâmica, em que cada elemento é criado com malloc. Nessas pilhas, é preciso liberar manualmente cada nó para evitar vazamentos de memória. Já nas pilhas com tamanho fixo (usando arrays), não se usa free, pois a memória é gerenciada de maneira automática. Para "limpar" a pilha estática, basta redefinir o topo para -1. Isso torna os elementos antigos inacessíveis pela lógica da pilha.

Considerações finais sobre o uso de pilhas

As operações de inserir (**push**), remover (**pop**), consultar (**peek**) e liberar (**free**) compõem o núcleo funcional da pilha. Com elas, é possível controlar com precisão a entrada e saída de dados, em contextos em que a ordem reversa é necessária.

Atenção

O uso correto das funções inserir, remover, consultar e liberar permite que a pilha se comporte de forma segura e previsível. Implementar pilhas de forma correta é fundamental para sistemas que exigem controle de histórico, processamento recursivo, análise de expressões ou controle de navegação.

Dominar a manipulação de pilhas torna o programador mais apto a resolver grande variedade de problemas algorítmicos e estruturais, fortalecendo sua base na programação estruturada e na lógica computacional.

Declaração, implementação e uso de pilhas

O vídeo a seguir ensina como implementar pilhas com vetores e struct em C. Ele demonstra também, com exemplos práticos, como inicializar a estrutura, manipular dados e integrar a estrutura ao menu principal do programa. Confira!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Fundamentos e criação de pilhas

Vamos agora aprender como declarar, implementar e utilizar uma pilha de Pessoa em linguagem C, utilizando vetores e uma estrutura de controle baseada no índice do topo.

A proposta é apresentar uma explicação detalhada de cada função envolvida no gerenciamento da pilha, desde a sua criação até operações clássicas, como inserção (**push**), remoção (**pop**) e consulta ao topo (**peek**).

Assim como já feito com filas, queremos aqui facilitar seu entendimento sobre a estrutura de pilha por meio de exemplos práticos com registros do tipo **Pessoa**, o que ajuda a visualizar como essas operações funcionam em aplicações reais. A implementação é feita passo a passo, com explicações completas para cada função do código.

Cada tópico, a seguir, irá mostrar uma parte de um código; o conjunto resultará na implementação completa de uma pilha, etapa por etapa. Mas primeiro, vamos entender melhor a estrutura e o que ela precisa para funcionar.

Declaração da estrutura

A pilha será implementada com uso de vetor fixo e uma variável que representa o índice do topo. Além disso, definimos um tipo **Pessoa** para armazenar elementos mais realistas, como podemos ver a seguir.

```c
#include
#include

#define MAX 5

typedef struct {
    char nome[30];
    int idade;
} Pessoa;

typedef struct {
    Pessoa itens[MAX];
    int topo;
} Pilha;
```

Explicação:

- **MAX:** define a capacidade máxima da pilha.
- **Pessoa:** é um tipo personalizado contendo nome e idade.
- **Pilha:** contém um vetor de Pessoa e um inteiro topo, que representa o índice do último elemento inserido. Quando a pilha está vazia, o topo é -1.

Inicialização e verificação

A pilha deve ser inicializada com o valor de **topo = -1**, indicando que está vazia. Veja!

```c
void inicializarPilha(Pilha *p) {
    p->topo = -1;
}
```

Explicação:

- Tal procedimento prepara a pilha para uso.
- A atribuição de **-1** ao topo é uma convenção que facilita a verificação de vazios e inserções futuras.

A seguir, verificamos se a pilha está **vazia**, ou seja, se não há elementos inseridos. Vamos lá!

```c
int pilhaVazia(Pilha *p) {
    return p->topo == -1;
}
```

Explicação:

- O valor retorna verdadeiro (1) se o topo for igual a -1.
- A verificação evitar remoções em uma pilha já vazia, o que causaria erro.

A seguir, buscamos se o **número máximo de elementos** já foi atingido.

```
c
int pilhaCheia(Pilha *p) {
    return p->topo == MAX - 1;
}
```

Explicação:

- O valor retorna verdadeiro (1) se o topo estiver no último índice possível do vetor.
- A verificação evita inserções que ultrapassariam o tamanho permitido da pilha.

Inserir (push)

Adiciona um novo elemento no topo da pilha. Confira!

```
c
void push(Pilha *p, Pessoa nova) {
    if (pilhaCheia(p)) {
        printf("Pilha cheia. Não é possível inserir.\n");
        return;
    }

    p->topo++;
    p->itens[p->topo] = nova;
}
```

Explicação:

- O método verifica se a pilha está cheia.
- Em caso negativo, o método incrementa o topo e insere o novo elemento naquela posição.
- Push é uma operação rápida e eficiente, com complexidade constante (O(1)).

Remover (pop)

Retira o elemento do topo da pilha, como podemos ver adiante:

```c
void pop(Pilha *p, Pessoa *removida) {
    if (pilhaVazia(p)) {
        printf("Pilha vazia. Não é possível remover.\n");
        return;
    }

    *removida = p->itens[p->topo];
    p->topo--;
}
```

Explicação:

- O método verifica se a pilha está vazia antes de tentar remover.
- A função copia o conteúdo do topo para o ponteiro removido, e então reduz o topo.
- Isso simula bem o comportamento LIFO (Last In, First Out).

Consulta (peek)

Veja, a seguir, como o método **peek** consulta o elemento no topo sem removê-lo.

```c
void peek(Pilha *p, Pessoa *visualizada) {
    if (pilhaVazia(p)) {
        printf("Pilha vazia. Nada para espiar.\n");
        return;
    }

    *visualizada = p->itens[p->topo];
}
```

Explicação:

- A função é muito útil quando queremos saber quem é o último elemento inserido, mas sem o alterar.
- O método evita mudanças na estrutura da pilha, mantendo o topo inalterado.

Mostar pilha

Possibilita visualizar todos os elementos da pilha, do topo até a base. Acompanhe!

```c
void mostrarPilha(Pilha *p) {
    printf("Pilha (topo -> base):\n");
    for (int i = p->topo; i >= 0; i--) {
        printf("[%s, %d]\n", p->itens[i].nome, p->itens[i].idade);
    }
    printf("\n");
}
```

Explicação:

- A visualização começa do topo e vai até o primeiro elemento inserido.
- A função é útil para depuração ou apresentação dos dados empilhados.

Função main e utilização

Neste exemplo, temos a utilização da função main() para executar toda a estrutura criada e apresentada nos tópicos anteriores. Caso queira, faça modificações para experimentar e trabalhar o entendimento sobre a estrutura.

```
c

int main() {
    Pilha p;
    inicializarPilha(&p);

    Pessoa a = {"Ana", 20};
    Pessoa b = {"Bruno", 35};
    Pessoa c = {"Carlos", 28};

    push(&p, a);
    push(&p, b);
    push(&p, c);

    mostrarPilha(&p);

    Pessoa removida;
    pop(&p, &removida);
    printf("Removida: %s, %d\n", removida.nome, removida.idade);

    mostrarPilha(&p);

    Pessoa topo;
    peek(&p, &topo);
    printf("Topo atual: %s, %d\n", topo.nome, topo.idade);

    return 0;
}
```

Explicação:

- A função inicializa a pilha e empilha três pessoas.
- O método remove o elemento do topo e mostra novamente a pilha.
- A função usa **peek** para inspecionar o novo topo sem o alterar.

Compreender e saber implementar pilhas é essencial para quem estuda estruturas de dados. Elas são utilizadas em problemas como:

- Controle de chamadas de função (pilha de execução).

- Navegação com desfazer/refazer.
- Conversão de expressões matemáticas (notação pós-fixada).
- Avaliação de expressões aritméticas.
- Backtracking (busca de soluções).

Cada função (a saber: **push**, **pop** e **peek**) representa um passo importante na manipulação da pilha. Juntas, elas garantem o funcionamento correto da estrutura LIFO.

Manter a verificação de limites (cheia ou vazia) é indispensável para que o programa esteja muito bem estruturado e livre de erros de acesso inválido.

Hora de codar

Veja, neste vídeo, a implementação de uma pilha em C de Pessoas, essencial para criar a pilha das peças no Tetris Stack.

O vídeo aplica os conceitos da pilha em um exercício prático, no qual é possível montar e manipular uma pilha de Pessoas. Imperdível!

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

Desafio: nivel aventureiro

O vídeo a seguir convida você a combinar o uso da fila com a pilha de reserva. Assista!

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

O que você vai fazer

Implementar um programa em C que simula o gerenciamento de peças no jogo Tetris Stack, utilizando **uma fila circular** de peças futuras e **uma pilha de reserva**. Cada peça tem um tipo (como 'I', 'O', 'T', 'L') e um identificador numérico único (id). O jogador poderá visualizar a fila e a pilha, jogar peças, reservar peças para uso posterior e remover peças da reserva.

> pilha de reserva
>
> A pilha de reserva não é um novo tipo de estrutura de dados, mas o papel estratégico que a pilha desempenha no jogo. Seu objetivo é que o jogador guarde a peça atual para usá-la em um momento mais oportuno.

Requisitos funcionais

Seu programa em C deverá:

- **Inicializar a fila de peças** com um número fixo de elementos (por exemplo, 5).
- **Inicializar uma pilha de peças** reservadas com capacidade limitada (por exemplo, 3).

Seu programa em C também deverá permitir as seguintes ações:

- **Jogar uma peça**: remove a peça da frente da fila (dequeue).
- **Reservar uma peça**: move a peça da frente da fila para o topo da pilha, se houver espaço.
- **Usar uma peça reservada**: remove a peça do topo da pilha, simulando seu uso.
- **Exibir o estado atual**: mostra as peças na fila e na pilha após cada ação.

Atenção: as peças removidas da fila ou da pilha não voltam para o jogo.

A cada ação, uma nova peça é automaticamente gerada e adicionada ao final da fila, mantendo-a sempre cheia.

Atributos das peças

Cada peça possui:

- **nome**: caractere que representa o tipo da peça ('I', 'O', 'T', 'L').
- **id**: número inteiro único que representa a ordem de criação da peça.

As peças são geradas automaticamente por uma função chamada **gerarPeca**.

Exemplo de saída

Estado atual:

Fila de peças	[T 0] [O 1] [L 2] [I 3] [I 4]
Pilha de reserva	(Topo → Base): [T 7] [O 6]

Tabela: Visualização atual da fila de peças e da pilha de reserva.
Curadoria de TI.

Opções de Ação:

Código	Ação
1	Jogar peça
2	Reservar peça
3	Usar peça reservada

0	Sair

Tabela: Comandos para movimentar peças entre a fila e a pilha de reserva.
Curadoria de TI.

Opção: ___

Requisitos não funcionais

Observe os seguintes elementos importantes:

- **Usabilidade**: a saída do programa deve ser clara e fácil de entender, com separação visual entre fila e pilha.
- **Legibilidade**: o código deve ser bem organizado, com comentários explicando a lógica utilizada. Utilize nomes de variáveis descritivos.
- **Documentação**: comente seu código, explicando o propósito de cada parte.

Simplificações para o nível intermediário

O foco aqui está no uso combinado da fila e da pilha. Algumas limitações são:

- O jogador **não pode escolher o tipo da peça,** pois elas são sempre geradas de forma aleatória.
- Impossibilidade de trocar diretamente a peça da fila com a da pilha.
- A fila sempre mantém o tamanho, e a pilha tem um tamanho máximo fixo.
- O menu é simples, com quatro opções fixas: jogar, reservar, usar peça da reserva e sair.

Conceitos trabalhados

Os pontos fundamentais são:

- **Fila circular:** manipulação eficiente de elementos com reaproveitamento de espaço.
- **Pilha linear**: armazenamento em estilo LIFO (último a entrar, primeiro a sair).
- **Structs e arrays:** definição e uso de tipos personalizados para representar peças.
- **Entrada e saída de dados:** interação com o jogador via terminal.
- **Funções e modularização:** separação de responsabilidades no código.
- **Operadores lógicos e condicionais:** controle de fluxo para validação de operações e restrições.

Entregando seu projeto

1. **Desenvolva seu projeto no GitHub:** use o mesmo repositório dos níveis anteriores.
2. **Atualize o arquivo do seu código:** atualize o arquivo super_trunfo.c com o código completo, incluindo as novas funcionalidades.

3. **Compile e teste:** faça isso de modo rigoroso, garantindo que todas as comparações e cálculos estejam corretos.

4. **Faça commit e push:** faça commit das suas alterações e envie (push) para o seu repositório no GitHub.

5. **Envie o link do repositório no GitHub:** faça isso por meio da plataforma SAVA.

Tutorial git

Você está prestes a aplicar os conceitos aprendidos para resolver um desafio prático no ambiente do GitHub. Veja as instruções gerais a seguir para acessar, aceitar e executar o desafio, garantindo que sua solução esteja bem estruturada e documentada.

Dê o primeiro passo

Acesse o GitHub Classroom. Nesse ambiente, você terá acesso ao repositório padrão do desafio. Caso ainda não tenha uma conta no GitHub, não se preocupe: você pode criar uma grátis, clicando no link.

Aceite o desafio

Tenha acesso ao repositório no GitHub, no qual você encontrará o repositório criado para o desenvolvimento do seu desafio.

Acesse o repositório

Clique no link do repositório para abrir o ambiente GitHub com a descrição do desafio e a estrutura modelo de arquivos e pastas que deve ser utilizada. Lembre-se: é esse o link que você deve enviar no SAVA.

Explore a estrutura do ambiente

Veja a estrutura organizada de pastas e arquivos necessários para o desenvolvimento do desafio.

Desenvolva o desafio

Utilize o GitHub CodeSpace para editar o arquivo do código-fonte e desenvolver o desafio. Certifique-se de que o código esteja organizado e funcional para resolver o problema proposto.

Entregue o desafio

Forneça o repositório do GitHub com todos os arquivos de código-fonte e conteúdos relacionados ao projeto. Certifique-se de que o repositório esteja bem estruturado, com pastas e arquivos nomeados de maneira clara e coerente. Envie o link para o repositório do seu desafio no GitHub.

Por fim, comente todos os arquivos de código-fonte, pois isso ajuda a demonstrar o quanto você sabe sobre o funcionamento do código e facilita a correção por terceiros. Assim, seus comentários devem explicar a

finalidade das principais seções do código, o funcionamento de algoritmos complexos e o propósito de variáveis e funções utilizadas.

Veja o vídeo a seguir, que traz dicas e detalhes sobre a entrega do seu projeto!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução à integração a filas e pilhas

O vídeo a seguir explica por que integrar estruturas como pilha e fila, destacando suas diferenças e como podem ser combinadas em sistemas complexos. Assista!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Explorando pilhas e filas: fundamentos e integração

No universo da programação, o domínio de estruturas de dados é um dos pilares para a construção de soluções eficientes, robustas e organizadas. Entre essas estruturas, **pilhas** e **filas** possuem suas regras específicas de inserção e remoção de dados.

A pilha e fila são estudadas de forma isolada, ajudando a compreender seus comportamentos e limitações. No entanto, à medida que os desafios da programação se tornam mais complexos, cresce a necessidade de compreender **como essas duas estruturas podem se comunicar e funcionar em conjunto**, formando sistemas mais dinâmicos e realistas.

Importância da integração entre pilhas e filas

É devido ao fato de ampliar as possibilidades de organização e manipulação de dados em aplicações reais. Muitas vezes, situações do mundo real não podem ser modeladas por uma única estrutura isolada. Assim, ao combinar pilhas e filas, o programador passa a ter mais flexibilidade para representar fluxos de dados, implementar regras de prioridade, administrar reservas temporárias de elementos ou mesmo simular comportamentos estratégicos em jogos e sistemas operacionais.

Esse tipo de combinação é utilizado em:

Sistemas de atendimento

Há call centers ou chats de suporte utilizam filas para organizar as solicitações dos clientes e pilhas para armazenar as últimas interações com o cliente.

Sistemas de navegação

Armazenam os comandos do usuário em fila para não perderem a ação em casos de lentidão. Ao mesmo tempo, eles transferem esses valores para a pilha, para serem utilizados no sistema de desfazer.

Sistemas de cache

Combinam pilhas para gerenciar elementos de acesso recente e filas para descartar elementos antigos conforme novos chegam.

Entender como fazer essa integração é mais do que só saber como copiar ou transferir elementos de uma estrutura para outra. Trata-se de projetar uma **comunicação consciente** entre estruturas que operam de formas opostas, respeitando suas particularidades e limites. Essa comunicação pode envolver a movimentação de dados entre a pilha e a fila, trocas parciais ou totais, sincronização de acessos, controle de estado e até mesmo a imposição de regras intermediárias que só se tornam viáveis por meio do uso coordenado dessas estruturas.

Benefícios

Ao integrar essas estruturas, o programador passa a desenvolver **habilidades de abstração e modelagem mais avançadas**. Ele deixa de pensar apenas na estrutura isolada e começa a enxergar o comportamento do sistema como um todo, sendo capaz de antecipar problemas, planejar soluções escaláveis e organizar o fluxo de informações de forma mais coesa. Esse tipo de raciocínio é essencial em áreas como inteligência artificial, em que os agentes precisam reagir a estímulos em tempo real, ou mesmo em sistemas embarcados, nos quais a comunicação entre diferentes módulos exige estruturas que possam cooperar entre si.

Outro benefício da integração entre pilhas e filas é a **capacidade de alternar perspectivas temporais**, técnica muito utilizada em sistemas web e mobile, nos quais a pilha armazena páginas anteriores e a fila, as futuras. Enquanto a fila lida com a ordem cronológica natural dos eventos, a pilha oferece uma forma de resgatar o último estado armazenado.

Portanto, a partir da coexistência integração entre esses dois mecanismos cria-se um ambiente computacion capaz de tratar tanto processos sequenciais como situações emergenciais ou contextuais.

Encerramento: prática e visão na programação

O aprendizado sobre essa comunicação não se resume apenas à teoria. Ele é reforçado por práticas que envolvem **a construção de menus interativos**, **simulações** e **desafios que exigem manipulação dinâmica de dados**. Tais práticas mostram que a programação vai além da codificação de comandos; ela envolve planejamento, estratégia e adaptação. Saber integrar pilhas e filas com clareza, segurança e propósito é um diferencial importante para qualquer desenvolvedor que queira criar sistemas bem estruturados e preparados para lidar com diferentes tipos de demandas.

> Compreender e aplicar a comunicação entre pilhas e filas é uma questão de visão computacional ampliada, de modo que o programador transite com mais liberdade entre os diferentes contextos e cenários do desenvolvimento.

É no encontro entre ordem e reversibilidade, fluxo contínuo e armazenamento temporário, que surgem soluções mais criativas, eficientes e alinhadas aos desafios reais da computação.

Comunicação e integração entre fila e pilha

Veja o vídeo a seguir, que mostra, por meio de exemplos práticos, como transferir dados de uma fila para uma pilha com base em critérios (como idade).

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Integração entre pilhas e filas: primeiros passos

Na construção de algoritmos eficientes, muitas vezes, é necessário ir além do uso isolado de estruturas de dados. Um dos primeiros passos é compreender como diferentes estruturas podem **se comunicar** e **se integrar** para atingir objetivos comuns. No caso da fila (estrutura FIFO) e da pilha (estrutura LIFO), essa integração abre espaço para soluções criativas e práticas, mesmo em sistemas simples.

Vamos agora entender como podemos usar fila e pilha juntas para processar, filtrar ou redirecionar dados de forma condicional. Para isso, utilizaremos como exemplo uma fila de pessoas contendo nome e idade, em que algumas pessoas serão transferidas para uma pilha com base em uma regra: idade maior que 60 anos.

Aplicação

Quando dizemos que estruturas estão integradas, afirmamos que elas **compartilham ou transferem dados entre si**. No caso da fila e da pilha, essa transferência pode ser feita com base em alguma lógica específica, como:

- Redirecionar dados de acordo com um filtro (ex: idade).
- Modificar a ordem de processamento.
- Separar tipos distintos de dados em fluxos diferentes.

Vamos imaginar um cenário em que estamos organizando o atendimento em um hospital. Pacientes chegam e entram em uma **fila de espera**, mas, se forem **idosos**, são direcionados para uma **pilha de prioridade**.

Isso exige:

- Percorrer a fila
- Analisar cada elemento (idade)
- Redirecionar ou manter o dado
- Preservar a integridade das estruturas

Se aplicado da forma correta, a integração entre pilhas e filas pode ser útil em situações como: sistema de triagem médica, separação de prioridades e filtragem por critérios, entre outros.

Exemplo

Neste caso, temos uma aplicação prática da integração entre fila e pilha para filtrar elementos com base em uma condição. A estrutura da fila é utilizada para armazenar uma sequência de pessoas, e a pilha funciona como um espaço reservado para aquelas que atendem a um critério específico (neste caso, pessoas com mais de 60 anos).

Toda a implementação das estruturas é mantida conforme o que estudamos: temos os **structs** de Pessoa, **Fila** e **Pilha**, além das funções como **inserir**, **remover**, **push**, **pop** e **peek**. A diferença está na lógica implementada dentro da função **main()**, na qual ocorre o verdadeiro processo de integração entre as estruturas. Vamos lá!

c

```
#include
#include
#include

#define MAX 50

typedef struct {
    char nome[30];
    int idade;
} Pessoa;

typedef struct {
    Pessoa dados[MAX];
    int inicio, fim;
} Fila;

typedef struct {
    Pessoa dados[MAX];
    int topo;
} Pilha;

void inicializarFila(Fila *f) {
    f->inicio = 0;
    f->fim = 0;
}

void inicializarPilha(Pilha *p) {
    p->topo = -1;
}

void inserir(Fila *f, Pessoa p) {
    if (f->fim < MAX) {
        f->dados[f->fim++] = p;
    }
}

Pessoa remover(Fila *f) {
    return f->dados[f->inicio++];
}

void push(Pilha *p, Pessoa pes) {
    if (p->topo < MAX - 1) {
        p->dados[++p->topo] = pes;
    }
}

Pessoa pop(Pilha *p) {
    return p->dados[p->topo--];
}

Pessoa peek(Pilha *p) {
    return p->dados[p->topo];
}

void exibirPilha(Pilha p) {
    while (p.topo >= 0) {
        Pessoa atual = pop(&p);
        printf("Nome: %s | Idade: %d\n", atual.nome, atual.idade);
    }
}

int main() {
    Fila fila;
    Pilha pilhaPrioridade;

    inicializarFila(&fila);
```

A lógica é a seguinte: os elementos são inseridos na fila como de costume. Depois, ela é percorrida, e cada elemento é removido e avaliado. Se a pessoa tiver idade acima de 60 anos, ela é enviada para a pilha com a função **push**. Ao final, a pilha contém apenas os elementos filtrados, e sua exibição reflete o comportamento LIFO típico dessa estrutura.

Esse exemplo demonstra como a combinação de estruturas diferentes organiza, separa e processa dados com mais controle, possibilitando aplicar filtros e manipulações específicas sem comprometer a integridade dos dados originais. É uma estratégia útil para organização de prioridades, triagens e classificações em sistemas reais.

Resumindo conceitos de integração entre fila e pilha

A integração entre fila e pilha é uma ferramenta importante na modelagem de problemas reais. Saber como transferir dados de forma controlada entre as duas estruturas amplia a capacidade de quem programa de construir sistemas adaptáveis, eficientes e organizados. Exercícios com filtragem de dados e redirecionamento de fluxos são ideais para consolidar esses conceitos.

Reversão de ordem de uma fila

Confira, neste vídeo, uma aplicação clássica da integração entre pilha e fila: inverter a ordem dos elementos de uma fila. Veja ainda como usar a pilha como estrutura auxiliar para reordenar os dados de forma eficiente.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Uso de pilha para inverter uma fila

Reverter a ordem de elementos em uma fila é uma tarefa frequente em algoritmos. Embora a fila por si só não ofereça um meio direto para isso, a pilha pode ser usada como estrutura auxiliar para **inverter a ordem** dos dados. Essa técnica é simples, poderosa e excelente para praticar a comunicação entre estruturas com comportamentos opostos.

Vamos agora usar **fila** e **pilha juntas** para implementar essa inversão de forma didática e estruturada.

Aplicação

A lógica da reversão é composta por dois passos:

Transferir os elementos da fila para a pilha

Como a pilha inverte a ordem de forma natural, o último da fila será o topo da pilha.

Transferir os elementos da pilha de volta para outra fila

Como resultado, temos uma nova fila com os elementos na ordem invertida

A técnica da aplicação é útil em vários contextos, como desfazer ações, reorganizar comandos, interpretar instruções reversas e até gerar saídas específicas de fluxos.

Exemplo

Vamos à seguinte situação prática, que demonstra a integração entre uma fila e uma pilha para inverter a ordem dos elementos de uma fila. Vamos manter todas as estruturas e funções de operação de cada tipo de estrutura de dados (fila e pilha), como inserir (enqueue), remover (dequeue), push e pop, além das funções de inicialização e exibição.

c

```
#include
#include
#include

#define MAX 50

typedef struct {
    char nome[30];
    int idade;
} Pessoa;

typedef struct {
    Pessoa dados[MAX];
    int inicio, fim;
} Fila;

typedef struct {
    Pessoa dados[MAX];
    int topo;
} Pilha;

void inicializarFila(Fila *f) {
    f->inicio = 0;
    f->fim = 0;
}

void inicializarPilha(Pilha *p) {
    p->topo = -1;
}

void inserir(Fila *f, Pessoa p) {
    if (f->fim < MAX) {
        f->dados[f->fim++] = p;
    }
}

Pessoa remover(Fila *f) {
    return f->dados[f->inicio++];
}

void push(Pilha *p, Pessoa pes) {
    if (p->topo < MAX - 1) {
        p->dados[++p->topo] = pes;
    }
}

Pessoa pop(Pilha *p) {
    return p->dados[p->topo--];
}

void exibirFila(Fila f) {
    for (int i = f.inicio; i < f.fim; i++) {
        printf("Nome: %s | Idade: %d\n", f.dados[i].nome, f.dados[i].idade);
    }
}

int main() {
    Fila filaOriginal, filaInvertida;
    Pilha auxiliar;

    inicializarFila(&filaOriginal);
    inicializarFila(&filaInvertida);
    inicializarPilha(&auxiliar);

    Pessoa pessoas[] = {
        {"Carlos", 45}, {"Joana", 62}, {"Lucas", 30}, {"Maria", 75}
    }
```

A lógica principal também ocorre dentro da função **main()**, na qual é aplicada a comunicação entre as estruturas. Primeiro, os elementos são inseridos em uma fila na ordem original. Em seguida, todos os elementos dessa fila são removidos (dequeue) um a um e empilhados (push) em uma pilha auxiliar. Como a pilha segue o comportamento LIFO, ao transferir novamente os elementos da pilha para uma nova fila (filaInvertida), conseguimos inverter a ordem inicial dos dados.

Esse tipo de integração é bastante útil quando se deseja reorganizar dados de forma reversa, utilizando o comportamento natural da pilha como ferramenta auxiliar. A estrutura da fila permanece organizada e a pilha entra como apoio temporário, reforçando o conceito de que o uso combinado de diferentes estruturas ajuda a resolver problemas de forma mais eficiente e flexível.

Resumindo a reversão de filas com pilhas

Trata-se de uma estratégia simples, mas eficaz, que reforça o entendimento sobre o funcionamento interno dessas duas estruturas. Ao explorarmos a lógica da pilha (LIFO) em contraste com a fila (FIFO), conseguimos transformar a sequência de elementos de forma controlada e eficiente.

Além de ajudar a fixar os conceitos fundamentais de acesso e remoção de dados em estruturas diferentes, essa técnica também introduz padrões úteis na construção de algoritmos que exigem manipulação de ordem, como sistemas de desfazer/refazer, histórico de navegação ou reorganização de prioridades.

Portanto, dominar essa abordagem proporciona mais flexibilidade no desenvolvimento de soluções lógicas e prepara você para desafios mais complexos, envolvendo múltiplas estruturas em conjunto.

Hora de codar

Neste vídeo, acompanhe a implementação de uma fila circular com uma pilha em C de pessoas, que gera trocas e reversão de ordem.

Essa prática é usada para criar o sistema que realiza troca de peças entre a fila e a pilha no desafio, orienta a implementação de soluções integradas e demonstra reversão de dados e trocas controladas. Imperdível!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nivel mestre

Veja, no vídeo a seguir, como implementar trocas entre fila e pilha, permitindo não apenas jogar e reservar peças, mas também realizar trocas pontuais ou totais entre as estruturas.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O que você vai fazer

Desenvolver um gerenciador de peças que usa duas estruturas de dados: uma fila circular com capacidade para 5 peças e uma pilha com capacidade para 3 peças. O sistema irá executar ações estratégicas com elas, como jogar, reservar, recuperar e realizar uma troca em bloco entre as peças da fila e as da pilha.

> pilha de reserva
>
> A pilha de reserva não é um novo tipo de estrutura de dados, mas o papel estratégico que a pilha desempenha no jogo. Seu objetivo é que o jogador guarde a peça atual para usá-la em um momento mais oportuno.

Requisitos funcionais

Seu programa em C deverá:

- **Inicializar a fila de peças** com um número fixo de elementos (por exemplo, 5).
- **Inicializar uma pilha de peças** reservadas com capacidade limitada (por exemplo, 3).

Permitir as seguintes ações:

- **Jogar uma peça**: remove a peça da frente da fila (dequeue).
- **Reservar uma peça**: move a peça da frente da fila para o topo da pilha, se houver espaço.
- **Usar uma peça reservada**: remove a peça do topo da pilha, simulando seu uso.
- **Exibir o estado atual**: mostra as peças na fila e na pilha após cada ação.
- **Trocar peça atual:** substitui a peça da frente da fila com o topo da pilha.
- **Troca múltipla:** alterna as três primeiras peças da fila com as três peças da pilha (caso ambas tenham, pelo menos, 3 peças).
- Visualizar o estado atual da fila e da pilha.
- Gerar uma nova peça de forma automática a cada remoção ou envio à pilha, a fim de manter a fila sempre cheia (quando possível).
- Encerrar o programa.

Lembre-se: as peças removidas da fila ou da pilha não voltam para o jogo.

Atributos das peças

Cada peça possui:

- **nome**: caractere que representa o tipo da peça ('I', 'O', 'T', 'L').
- **id**: número inteiro único que representa a ordem de criação da peça.

As peças são geradas automaticamente por uma função chamada **gerarPeca**.

Exemplo de saída

Estado atual:

Fila de peças	[I 0] [L 1] [T 2] [O 3] [I 4]
Pilha de reserva	(Topo → base): [O 8] [L 7] [T 6]

Tabela: Visualização atual da fila de peças e da pilha de reserva.
Curadoria de TI.

Opções disponíveis:

Código	Ação
1	Jogar peça da frente da fila
2	Enviar peça da fila para a pilha de reserva
3	Usar peça da pilha de reserva
4	Trocar peça da frente da fila com o topo da pilha
5	Trocar os 3 primeiros da fila com as 3 peças da pilha
0	Sair

Tabela: Comandos para movimentar peças entre a fila e a pilha de reserva.
Curadoria de TI.

Opção escolhida: 5

Ação: troca realizada entre os 3 primeiros da fila e os 3 da pilha.

Novo estado:

Fila de peças	[O 8] [L 7] [T 6] [O 3] [I 4]
Pilha de reserva	(Topo → base): [T 2] [L 1] [I 0]

Tabela: Novo estado após a troca entre fila e pilha de peças.
Curadoria de TI.

Requisitos não funcionais

Observe os seguintes elementos importantes:

- **Usabilidade:** a saída do programa deve ser clara e fácil de entender, com separação visual entre fila e pilha.
- **Legibilidade:** o código deve ser bem organizado, com comentários explicando a lógica utilizada. Lembre-se: utilize nomes descritivos de variáveis.
- **Documentação:** comente seu código, explicando o propósito de cada parte.

Simplificações para o nível avançado

O foco é o uso combinado da fila e da pilha. Algumas limitações são:

- O jogador **não pode escolher o tipo da peça,** pois elas são sempre geradas de forma aleatória.
- Possibilidade de trocar diretamente a peça da fila com a da pilha.
- A fila sempre mantém o tamanho, e a pilha tem um tamanho máximo fixo.

Conceitos trabalhados

Os pontos fundamentais são:

- **Fila circular:** manipulação eficiente de elementos com reaproveitamento de espaço.
- **Integração de estruturas:** troca de valores em estruturas compostas.
- **Pilha linear**: armazenamento em estilo LIFO (último a entrar, primeiro a sair).
- **Structs e arrays:** definição e uso de tipos personalizados para representar peças.
- **Entrada e saída de dados:** interação com o jogador via terminal.
- **Funções e modularização:** separação de responsabilidades no código.
- **Operadores lógicos e condicionais:** controle de fluxo para validação de operações e restrições.

Entregando seu projeto

1. **Desenvolva seu projeto no GitHub:** use o mesmo repositório do GitHub dos níveis anteriores.
2. **Atualize o arquivo do seu código:** atualize o arquivo super_trunfo.c com o código completo, incluindo as novas funcionalidades.
3. **Compile e teste:** faça isso com rigor, garantindo que todas as comparações e cálculos estejam corretos.
4. **Faça commit e push:** faça commit das suas alterações e envie (push) para o seu repositório no GitHub.
5. **Envie o link do repositório no GitHub:** faça isso através da plataforma SAVA.

Tutorial git

Você está prestes a aplicar os conceitos aprendidos para resolver um desafio prático no ambiente do GitHub. Veja as instruções gerais a seguir para acessar, aceitar e executar o desafio, de modo que sua solução esteja bem estruturada e documentada.

Dê o primeiro passo

Acesse o GitHub Classroom. Nesse ambiente, você terá acesso ao repositório padrão do desafio. Caso ainda não tenha uma conta no GitHub, não se preocupe: você pode criar uma grátis, clicando no link.

Aceite o desafio
Tenha acesso ao repositório no GitHub, no qual você encontrará o repositório criado para o desenvolvimento do seu desafio.

Acesse o repositório
Clique no link do repositório para abrir o ambiente GitHub com a descrição do desafio e a estrutura modelo de arquivos e pastas que deve ser utilizada. Lembre-se: é esse o link que você deve enviar no SAVA.

Explore a estrutura do ambiente
Veja a estrutura organizada de pastas e arquivos necessários para o desenvolvimento do desafio.

Desenvolva o desafio
Utilize o GitHub CodeSpace para editar o arquivo do código-fonte e desenvolver o desafio. Certifique-se de que o código esteja organizado e funcional para resolver o problema proposto.

Entregue o desafio
Forneça o repositório do GitHub com todos os arquivos de código-fonte e conteúdos relacionados ao projeto. Certifique-se de que o repositório esteja bem estruturado, com pastas e arquivos nomeados de maneira clara e coerente. Envie o link para o repositório do seu desafio no GitHub.

Finalizando nosso estudo, aqui está a última orientação: comente todos os arquivos de código-fonte, pois isso demonstra o quanto você sabe sobre o funcionamento do código e facilita a correção por terceiros. Seus comentários devem explicar a finalidade das principais seções do código, o funcionamento de algoritmos complexos e o propósito de variáveis e funções utilizadas.

Assista agora ao último vídeo, com dicas e detalhes sobre a entrega do seu projeto. Até a próxima!

Conteúdo interativo
Acesse a versão digital para assistir ao vídeo.

Considerações finais

Parabéns pela conquista!

Ao longo deste conteúdo, você explorou em profundidade o funcionamento de filas e pilhas, aprendeu como elas operam de forma individual e como podem se comunicar e se integrar para resolver problemas mais complexos. Também experimentou técnicas importantes, como a reversão de ordem, a troca entre estruturas e a organização de dados em memória, tanto com estruturas estáticas quanto com alocação dinâmica.

Você também praticou o raciocínio lógico, a modularização de código, a manipulação de dados com structs e o uso estratégico de funções, consolidando um repertório valioso para resolver situações reais de programação. A prática orientada por problemas (como no projeto do Tetris Stack) mostrou que, com poucos recursos bem aplicados, é possível criar soluções elegantes, funcionais e eficientes.

Você está aprendendo a programar, é certo, mas também a resolver problemas com clareza, lógica e organização. Continue explorando, testando e criando, pois é assim que se constrói conhecimento de verdade!

Continue praticando

Confira a seguir três aplicações práticas que combinam filas e pilhas em diferentes contextos.

- **Sistema de desfazer/refazer**
  Crie um editor de texto simples com operações de *desfazer* e *refazer* utilizando duas pilhas. Cada ação do usuário é registrada para ser revertida ou refeita.

- **Filtro de mensagens**
  Implemente um sistema de moderação que recebe mensagens em uma fila e envia para uma pilha de revisão caso contenham certas palavras-chave. É uma aplicação prática de triagem condicional.

- **Simulador de atendimento bancário**
  Implemente uma fila de atendimento com prioridades. Clientes idosos, por exemplo, podem ser redirecionados para uma pilha de prioridade, simulando a triagem em tempo real.

Referências

CORMEN, T. H. *et al.* **Algoritmos**: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.

MANZANO, J. A. N. G.; OLIVEIRA, J. F. de. **Algoritmos**. São Paulo: Érica, 2014.