Jogo de Batalha Naval

Você irá desenvolver um programa em C que simule a jogabilidade do jogo Batalha Naval, utilizando vetores e matrizes para definir a posição dos navios e das áreas de habilidades especiais.

Curadoria de TI

1. Itens iniciais

Objetivos

- Aplicar o conceito de vetores e matrizes em linguagem C para representar um tabuleiro de Batalha Naval simplificado.
- Aplicar estruturas de repetição aninhadas (loops) e matrizes em linguagem C para construir e exibir um tabuleiro de Batalha Naval.
- Aplicar condicionais dentro de loops aninhados em linguagem C para simular áreas de efeito de habilidades especiais em um tabuleiro de Batalha Naval.

Introdução

Imagine que você acaba de ser contratado pela **Oceanic Games**, uma empresa inovadora no desenvolvimento de jogos, para trabalhar em um novo projeto: uma versão moderna e turbinada do clássico Batalha Naval. Seu papel nessa equipe será crucial: você será o programador responsável por implementar parte da lógica do jogo, definindo o posicionamento dos navios no tabuleiro e a área de ação das novas habilidades especiais, utilizando para isso seus conhecimentos de vetores e matrizes na linguagem C.

No universo da programação de jogos – e em diversas outras áreas da Tecnologia da Informação – a capacidade de modelar e manipular dados usando essas estruturas é fundamental para criar experiências interativas e eficientes. Dominar vetores e matrizes é uma habilidade muito requisitada no mercado de trabalho, permitindo que você crie algoritmos otimizados para lidar com grandes volumes de informações, desenvolva interfaces gráficas complexas e construa a base para sistemas robustos e escaláveis.

Este desafio prático irá guiá-lo por uma jornada que simula essa experiência real de desenvolvimento. Você construirá o jogo Batalha Naval passo a passo, em três módulos com níveis de dificuldade crescente. No primeiro módulo (Novato), você aprenderá a representar o tabuleiro e posicionar navios simples utilizando vetores. No segundo módulo (aventureiro), o desafio aumentará: você utilizará matrizes e loops para posicionar mais navios, incluindo em diagonal. Finalmente, no terceiro módulo (Mestre), você implementará as habilidades especiais do jogo, utilizando condicionais dentro de loops aninhados para definir áreas de efeito complexas como cones, cruzes e octaedros.

A cada módulo, você estará aplicando seus conhecimentos de forma prática, construindo um projeto completo e se preparando para os desafios reais da profissão. Prepare-se para mergulhar no código e transformar a teoria em um jogo funcional!

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

Cenário

A **Oceanic Games** decidiu desenvolver um jogo de Batalha Naval com uma mecânica inovadora: além das regras tradicionais, o jogo incluirá um sistema de habilidades especiais que afetam áreas específicas do tabuleiro. Essas habilidades, com diferentes formatos de alcance (cone, cruz e octaedro), adicionam uma camada estratégica ao jogo, permitindo que os jogadores ataquem múltiplas posições de uma só vez ou criem defesas em áreas específicas. Como desenvolvedor, seu desafio é implementar essas habilidades utilizando vetores e matrizes para representar o tabuleiro e calcular as áreas de efeito. Você precisará utilizar seus conhecimentos de lógica de programação, estruturas de dados e linguagem C para garantir que as

habilidades funcionem corretamente e que o jogo ofereça uma experiência balanceada e divertida para os jogadores.

Lembre-se: a precisão na manipulação de vetores e matrizes será crucial para determinar o sucesso ou o fracasso de suas estratégias no jogo – e no desenvolvimento do projeto para a **Oceanic Games**!

Introdução a vetores e matrizes

Neste vídeo, falaremos sobre o que são vetores e matrizes, por que são importantes e a forma básica de declarar. Também apontaremos as principais áreas da programação que se beneficiam com o uso de vetores e matrizes.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Conceito de vetores e matrizes

Vetores e matrizes são estruturas de dados fundamentais em programação, permitindo a organização e manipulação eficiente de grandes conjuntos de dados. Essas estruturas são amplamente utilizadas em diversas áreas da computação, desde o desenvolvimento de software e jogos até a análise de dados e inteligência artificial. A principal característica de vetores e matrizes é que elas armazenam elementos do mesmo tipo, organizados de forma estruturada para facilitar o acesso e a manipulação.

Atenção

Embora os termos "array" e "vetor" sejam frequentemente usados como sinônimos, especialmente em C, neste material faremos uma distinção entre eles para facilitar o aprendizado.

O que são arrays (vetores)?

Em programação, um **array** é uma estrutura de dados que armazena uma coleção de elementos do mesmo tipo em locais de memória contíguos (lado a lado). **Arrays podem ter uma ou mais dimensões. Um array unidimensional é chamado de vetor.**

Um **vetor**, portanto, é uma sequência ordenada de elementos, onde cada elemento pode ser acessado diretamente por sua posição na sequência, chamada de **índice**.

Uma característica importante dos arrays (e, consequentemente, dos vetores em C) é que o primeiro elemento sempre está localizado no índice zero. Por exemplo, um array de inteiros com cinco posições terá índices de 0 a 4.

```
c

      int numeros[5] = {10, 20, 30, 40, 50};
```

Neste exemplo, numeros é um vetor que armazena cinco números inteiros. Para acessar o primeiro elemento (o número 10), usaríamos numeros[0]. Para acessar o terceiro elemento (o número 30), usaríamos numeros[2], e assim por diante.

O que são matrizes?

Uma **matriz** é um array multidimensional. A matriz mais comum é a bidimensional, que pode ser visualizada como uma tabela organizada em linhas e colunas. Assim como nos vetores, cada elemento em uma matriz pode ser acessado por seus índices, mas agora precisamos de dois índices: um para a linha e outro para a coluna.

```
c

      int tabela[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Neste exemplo, tabela é uma matriz 3×3. Para acessar o elemento na primeira linha e segunda coluna (o número 2), usaríamos tabela[0][1]. O elemento na terceira linha e terceira coluna (o número 9) seria acessado com tabela[2][2]. A capacidade de organizar dados em múltiplas dimensões torna as matrizes ferramentas poderosas para representar informações complexas, como imagens, tabelas e grafos.

A diferença entre vetor e matrizes pode ser resumida em:

Vetor		Matriz
É unidimensional.		Pode ter duas ou mais dimensões.

Importância de arrays e matrizes

A utilização de arrays e matrizes em programação oferece diversas vantagens significativas, tornando-as estruturas de dados essenciais:

Eficiência

Arrays permitem acesso direto a qualquer elemento utilizando seu índice. Essa operação de acesso é realizada em tempo constante, o que significa que a velocidade de acesso permanece a mesma, independentemente do tamanho do array. Isso os torna ideais para situações em que a velocidade de acesso aos dados é crítica. O mesmo princípio se aplica a matrizes, onde o acesso a um elemento específico usando seus índices de linha e coluna também é uma operação em tempo constante.

Organização

Arrays e matrizes proporcionam uma maneira organizada e lógica de armazenar e acessar dados. Essa organização facilita a implementação de algoritmos e a manipulação de informações, tornando o código mais claro, legível e fácil de manter. Em algoritmos de processamento de dados, a estruturação oferecida por essas estruturas impacta diretamente na eficiência e na clareza do código.

Memória contígua

Arrays armazenam dados em blocos contíguos de memória. Essa característica otimiza o desempenho do processador, especialmente o uso da memória cache. Quando dados que são frequentemente acessados em conjunto estão próximos fisicamente na memória, o processador pode acessá-los mais rapidamente, resultando em operações mais eficientes. Esse princípio, conhecido como localidade espacial, também se aplica a matrizes, já que elas também são armazenadas de forma contígua na memória.

Aplicações de arrays e matrizes

A versatilidade de arrays e matrizes se reflete em suas diversas aplicações práticas em programação:

1

Ciência de dados e estatística

Arrays são utilizados para armazenar e manipular grandes conjuntos de dados numéricos, servindo como base para análises estatísticas e modelos de aprendizado de máquina. Matrizes são essenciais para realizar operações complexas de álgebra linear, como cálculos matriciais e manipulação de vetores, frequentemente usados em análises estatísticas avançadas.

2

Desenvolvimento de jogos

No desenvolvimento de jogos, arrays são usados para armazenar informações como posições de objetos, status de jogadores e pontuações. Matrizes são frequentemente empregadas para representar o mapa do jogo ou o tabuleiro, permitindo o acesso e a manipulação eficiente dos elementos do jogo.

3

Processamento de imagens

Imagens digitais são representadas como matrizes de pixels, onde cada elemento da matriz corresponde a um pixel e contém informações sobre sua cor. Operações como aplicação de filtros, redimensionamento e transformações geométricas são realizadas por meio de manipulações matemáticas nessa matriz de pixels. Arrays podem ser usados para armazenar informações auxiliares no processamento de imagens, como histogramas de cores.

4

Simulações científicas

Matrizes desempenham um papel fundamental em simulações científicas, permitindo modelar fenômenos físicos e realizar cálculos numéricos complexos. Por exemplo, em simulações de fluidos, uma matriz pode representar o campo de velocidades em uma determinada região do espaço. Arrays podem ser usados para armazenar parâmetros ou resultados intermediários da simulação.

5 Álgebra linear e computações científicas

Matrizes são a base da álgebra linear e são amplamente utilizadas em computações científicas para resolver sistemas de equações, realizar transformações lineares e executar outras operações matemáticas essenciais em áreas como física, engenharia e computação gráfica. Arrays servem como blocos de construção para implementar as operações da álgebra linear em código.

Compreender arrays e matrizes é fundamental para qualquer programador, pois, essas estruturas de dados fornecem uma forma eficiente e organizada de armazenar, acessar e manipular dados, facilitando a implementação de algoritmos complexos e a resolução de problemas em diversas áreas da computação. Dominar o uso de arrays e matrizes é essencial para desenvolver soluções robustas, otimizadas e escaláveis, consolidando uma base sólida para o desenvolvimento de software e para a atuação em diferentes campos da ciência da computação. A escolha entre usar um array unidimensional (vetor) ou multidimensional (matriz) dependerá da natureza do problema e da forma como os dados precisam ser organizados e acessados.

Arrays em C

Neste vídeo, explicaremos o que são arrays e seus usos mais comuns, apresentaremos e comentaremos alguns exemplos para facilitar a compreensão. Afinal, entender arrays é fundamental para dominar o uso de vetores. Vamos lá!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O que são arrays?

Também conhecidos como vetores, são uma das estruturas de dados fundamentais na programação. Eles permitem armazenar múltiplos valores do mesmo tipo em uma única variável, facilitando a organização e manipulação desses dados. Cada elemento do array pode ser acessado diretamente pelo seu índice, que começa em zero.

Definição e inicialização de arrays

Para declarar um array em C, você precisa especificar o tipo dos elementos e o número de elementos que o array irá armazenar. Por exemplo, para declarar um array de inteiros com cinco elementos, usamos:

```
c

#include

int main() {
    int numeros[5];  // Declaração de um array de inteiros com 5 elementos
    return 0;
}
```

Inicialização de arrays

Os arrays podem ser inicializados no momento da declaração. A inicialização pode ser feita fornecendo uma lista de valores entre chaves “**{}**” confira!

```c
#include

int main() {
    int numeros[5] = {10, 20, 30, 40, 50};  // Inicialização do array com valores
    return 0;
}
```

Nesse exemplo, o array **numeros** é inicializado com os valores 10, 20, 30, 40 e 50. Cada valor é atribuído a um elemento do array, começando do índice 0 até o índice 4.

Acessando elementos de um array

Cada elemento de um array pode ser acessado diretamente pelo seu índice. Os índices dos arrays começam em 0. Vamos ver como acessar e imprimir elementos específicos de um array. Observe!

Conteúdo interativo

esse a versão digital para executar o código.

Explicação do código

Vamos analisar cada linha desse programa.

Declaração e inicialização

“int numeros[5] = {10, 20, 30, 40, 50};” declara um array de inteiros chamado numeros com cinco elementos e os inicializa com os valores fornecidos.

Acesso aos elementos

numeros[0] acessa o primeiro elemento do array, que é 10.
numeros[2] acessa o terceiro elemento do array, que é 30 .
numeros[4] acessa o quinto elemento do array, que é 50 .

Impressão dos elementos

“printf("O primeiro elemento é %d\n", numeros[0]);” imprime o valor do primeiro elemento.
“printf("O terceiro elemento é %d\n", numeros[2]);” imprime o valor do terceiro elemento.
“printf("O quinto elemento é %d\n", numeros[4]);” imprime o valor do quinto elemento.

Aplicações simples de arrays

Os arrays são usados para armazenar conjuntos de dados relacionados de maneira eficiente. A seguir, veremos algumas aplicações simples de arrays, demonstrando como eles podem ser úteis em diversas situações. Vejamos cada uma a seguir.

Armazenamento de notas de alunos

Arrays podem ser usados para armazenar as notas de um grupo de alunos, facilitando o cálculo de médias e a verificação de desempenho. Veja o código a seguir.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, um array **notas** armazena as notas de três alunos e as imprime.

Armazenamento de caracteres

Arrays também podem ser usados para armazenar caracteres, como letras ou palavras. Esse exemplo demonstra como criar e acessar um array de caracteres. Confira!

Conteúdo interativo

esse a versão digital para executar o código.

Aqui, um array **letras** armazena quatro caracteres e os imprime.

Arrays utilizando string

Arrays podem armazenar strings, permitindo a manipulação de conjuntos de palavras ou frases. Veja o exemplo!

Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, criamos um array de strings com três nomes e os imprimimos usando um loop for. Cada elemento do array é acessado e exibido com printf.

Dois pontos importantes a destacar nesse exemplo. Primeiro, você deve ter percebido que na declaração do array nomes não foi especificado o tamanho do array. Isso não é problema, porque como ele foi inicializado com três nomes, automaticamente ele é definido com tamanho 3.

O segundo ponto é que você deve ter percebido que aparece um asterisco (*) antes da palavra nomes. Isso é uma indicação de ponteiros e é necessário quando você quiser criar um array de strings. Ponteiros é um assunto que está fora do escopo deste conteúdo, mas foi apresentado como são criados array de strings.

Resumindo

Arrays são uma maneira eficiente de armazenar e acessar múltiplos valores relacionados usando uma única variável. Em C, os arrays são definidos com um tamanho fixo e seus elementos são acessados por índices. Compreender como declarar, inicializar e acessar arrays é essencial para manipular dados de forma estruturada e eficiente na programação. As aplicações de arrays são vastas, variando desde armazenar notas de alunos até armazenar caracteres para processamento de texto.

Matrizes em C

Neste vídeo, apresentaremos o que são vetores e matrizes, abordando-os separadamente e em conjunto. Em seguida, discutiremos práticas de uso e diferentes formas de aplicar essas estruturas.

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

Matrizes são estruturas de dados fundamentais na programação que permitem armazenar e manipular conjuntos de valores organizados em múltiplas dimensões. Elas são amplamente utilizadas para representar e organizar dados de forma eficiente. Vamos explorar o que são matrizes, como são definidas e manipuladas, fornecendo exemplos de código para ilustrar esses conceitos.

O que são matrizes?

São uma generalização dos arrays. Enquanto um array é unidimensional, uma matriz pode ter duas ou mais dimensões. A matriz bidimensional, a mais comum, é essencialmente uma tabela ou grade de valores em que cada elemento é acessado por dois índices: um para a linha e outro para a coluna.

Definição e inicialização de matrizes

Uma matriz bidimensional de inteiros pode ser definida da seguinte forma. Confira!

> Conteúdo interativo
>
> esse a versão digital para executar o código.

Nesse exemplo, **matriz** é uma matriz 3×3. Cada elemento é acessado usando dois índices, correspondendo à linha e à coluna.

Inicialização simplificada

Semelhante aos vetores, você pode inicializar uma matriz ao mesmo tempo em que a declara:

```
c

int matriz[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Acessando elementos de uma matriz

Cada elemento de uma matriz pode ser acessado diretamente pelos seus índices de linha e coluna. Vamos ver como acessar e imprimir elementos específicos de uma matriz. Veja!

> Conteúdo interativo
>
> esse a versão digital para executar o código.

No código apresentado, o programa em C define uma matriz 3×3 chamada matriz e preenche essa matriz com valores inteiros. A matriz é inicializada com os seguintes valores:

4 5 6

7 8 9

Cada elemento da matriz é acessado por meio de índices que especificam sua posição, onde o primeiro índice indica a linha e o segundo índice indica a coluna. Por exemplo, matriz[0][0] se refere ao elemento na primeira linha e primeira coluna.

Vamos ver o que acontece com cada printf.

printf("O elemento na posição [0][0] é %d\n", matriz[0][0]); Este comando acessa o elemento na primeira linha e primeira coluna da matriz, que é o número 1. Portanto, o programa imprime: 1

printf("O elemento na posição [1][1] é %d\n", matriz[1][1]); Aqui, o comando acessa o elemento na segunda linha e segunda coluna, que é o número 5. Assim, o programa imprime: 5

printf("O elemento na posição [2][2] é %d\n", matriz[2][2]); Neste caso, o comando acessa o elemento na terceira linha e terceira coluna, que é o número 9. Então, o programa imprime: 9

Considerações importantes

Ao trabalhar com matrizes em C, existem alguns pontos importantes que estão descritos a seguir.

Tamanho fixo

Em C, matrizes têm tamanho fixo determinado no momento da sua declaração. O tamanho não pode ser alterado em tempo de execução.

Acesso por índice

O acesso aos elementos é feito por índices, que começam em zero. É importante garantir que os índices utilizados estejam dentro dos limites do vetor ou matriz para evitar comportamento indefinido.

Tipagem

Todos os elementos de um vetor ou matriz são do mesmo tipo, seja int, float, char etc.

Matrizes são fundamentais para a organização e manipulação de dados na programação. Elas permitem o armazenamento de múltiplos valores em uma estrutura que facilita o acesso eficiente aos dados através de índices.

Compreender como definir, inicializar e acessar matrizes é essencial para resolver problemas complexos de forma estruturada e eficiente.

Hora de codar

Neste vídeo, aplicaremos os conceitos básicos de vetores e matrizes, explorando como declarar, inicializar e acessar seus elementos através de exemplos práticos. Você aprenderá a manipular dados eficientemente, utilizando estruturas de repetição, consolidando seu entendimento para resolver problemas de programação em C. Não deixe de assistir!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível novato

Posicionando Navios no Tabuleiro

Neste primeiro desafio, você dará o primeiro passo na construção do seu jogo de Batalha Naval. Você utilizará seus conhecimentos de vetores (arrays unidimensionais) em C para representar um tabuleiro simplificado e posicionar dois navios nele: um na vertical e outro na horizontal. Continue o desenvolvimento no mesmo programa iniciado anteriormente.

O que você vai fazer

1. **Represente o Tabuleiro:** Utilize uma matriz (array bidimensional) para representar o tabuleiro do Batalha Naval. Neste nível novato, o tabuleiro terá um tamanho fixo 10×10. Inicialize todas as posições do tabuleiro com o valor 0, representando água.

2. **Posicione os Navios:** Declare e inicialize dois vetores (arrays unidimensionais) para representar os navios. Cada navio ocupará um número fixo de posições no tabuleiro (defina esse tamanho, por exemplo, 3 posições). Um navio será posicionado horizontalmente e o outro verticalmente. Represente as posições ocupadas pelos navios na matriz do tabuleiro com o valor 3. Você deverá escolher as coordenadas iniciais de cada navio e garantir que eles estejam completamente dentro dos limites do tabuleiro e não se sobreponham.
   Dica: O posicionamento do navio pode ser feito copiando o valor 3 de cada posição do vetor do navio para as posições correspondentes na matriz do tabuleiro, de acordo com a orientação (horizontal ou vertical) do navio.

3. **Exiba o Tabuleiro:** Utilize loops aninhados e o comando printf para exibir o tabuleiro no console. Mostre a matriz completa, com 0s representando água e 3s representando as partes dos navios. A saída deve ser clara e organizada, permitindo visualizar facilmente a posição dos navios.
   Dica: Imprima um espaço ou outro caractere separador entre os elementos da matriz para facilitar a visualização.

Requisitos funcionais

- O programa deve receber as coordenadas iniciais (linha e coluna) de cada navio como entrada (pode ser definido diretamente no código).

- O programa deve validar se as coordenadas e o tamanho dos navios são válidos dentro dos limites do tabuleiro.

- O programa deve garantir que os navios não se sobreponham.

- O programa deve exibir o tabuleiro no console com os navios posicionados corretamente.

Requisitos não funcionais

- **Performance:** O programa deve executar de forma eficiente, sem causar atrasos perceptíveis.
- **Documentação:** O código deve ser bem documentado, com comentários claros explicando a lógica e o propósito de cada parte do programa. Utilize comentários para explicar a função de cada variável, loop e bloco de código.
- **Legibilidade:** O código deve ser escrito de forma clara, organizada e fácil de entender, com nomes de variáveis descritivos e indentação consistente.

Simplificações para o nível básico

- O tamanho do tabuleiro e dos navios é fixo (10×10).
- Os navios têm tamanho fixo igual a 3.
- As coordenadas dos navios são definidas diretamente no código, sem input do usuário.
- Não é necessário implementar a lógica do jogo (ataques, acertos, etc.) neste nível.
- A validação de sobreposição de navios pode ser simplificada.

Entregando seu Projeto

1. **GitHub:** Hospede seu projeto em um repositório no GitHub. Isso facilitará o versionamento do seu código e o compartilhamento com o professor.
2. **SAVA:** Envie o link do seu repositório GitHub na plataforma SAVA, na atividade correspondente ao Desafio Nível Básico. Certifique-se de que o repositório seja público ou que o professor tenha acesso a ele. O link do repositório deve ser **a única forma de entrega do projeto**, garantindo que o professor possa acessar seu código para avaliação.

Lembre-se: este desafio foca nos fundamentos de vetores e matrizes. Concentre-se em escrever um código claro, funcional e bem documentado, demonstrando sua compreensão dos conceitos aprendidos.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

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

Para começar, 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

Após aceitar a tarefa, você receberá acesso ao repositório no GitHub, nele 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. É esse link que você deve enviar no SAVA.

Explore a estrutura do ambiente

No ambiente do GitHub, você verá 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

Para a entrega, você precisará fornecer o repositório do GitHub que contém 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.

Comente todos os arquivos de código-fonte. Os comentários são indispensáveis para demonstrar seu entendimento sobre o funcionamento do código e facilitar a correção por terceiros. Eles devem explicar a finalidade das principais seções do código, o funcionamento de algoritmos complexos e o propósito de variáveis e funções utilizadas.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução a vetores e matrizes com loops

Neste vídeo, será abordado como funciona o uso de vetores e matrizes com a utilização de estruturas de repetição aninhadas. Também mostraremos as áreas mais beneficiadas da adoção dessa prática e seus benefícios de utilização.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Uso de matrizes e vetores com loops aninhados na programação

É uma técnica amplamente difundida e fundamental para a manipulação de grandes conjuntos de dados. Quando combinados com loops, esses elementos se tornam ainda mais poderosos, permitindo a realização de operações complexas de forma eficiente. A compreensão e a aplicação correta de vetores, matrizes e loops são habilidades essenciais para qualquer programador, independentemente da linguagem de programação utilizada.

Aplicações práticas

Apresentaremos agora diversas aplicações práticas de vetores e matrizes combinados com loops aninhados. Essas aplicações incluem a soma de matrizes, processamento de imagens, desenvolvimento de jogos e simulações, análise de dados e operações de álgebra linear.

Cada item a seguir irá explicar como essas estruturas de dados podem ser utilizadas para resolver problemas complexos de maneira eficiente e organizada, destacando a importância de dominar essas técnicas na programação. Vamos conferir!

Soma de matrizes

É uma operação básica que exemplifica bem o uso de loops aninhados. Cada elemento da matriz resultante é a soma dos elementos correspondentes das matrizes de entrada. Isso é feito com um loop aninhado, em que o loop externo percorre as linhas e o loop interno percorre as colunas, realizando a soma elemento a elemento.

Processamento de imagens

Uma imagem pode ser representada como uma matriz de pixels. Operações como a aplicação de filtros (blur, edge detection) envolvem a manipulação dos pixels vizinhos de cada ponto da imagem. Loops aninhados são usados para percorrer cada pixel e seus vizinhos, aplicando a operação desejada e atualizando a matriz da imagem.

Jogos e simulações

Em jogos de tabuleiro, como xadrez ou sudoku, e em simulações, como autômatos celulares, as matrizes representam o estado do jogo ou do sistema. Loops aninhados permitem atualizar o estado de cada célula do tabuleiro ou do sistema com base nas regras do jogo ou da simulação. Isso inclui verificação de condições de vitória, movimentação de peças e cálculo de próximas gerações em simulações.

Análise de dados

Matrizes são usadas para representar grandes conjuntos de dados tabulares na análise de dados, especialmente em big data. Operações como normalização, cálculo de médias e desvio-padrão e transformações lineares são realizadas através de loops aninhados que percorrem cada elemento da matriz, realizando as operações necessárias.

Álgebra linear e computações científicas

A álgebra linear é uma área na qual o uso de matrizes é predominante. Operações como multiplicação de matrizes, inversão e decomposições são realizadas usando loops aninhados. Por exemplo, a multiplicação de duas matrizes envolve três loops: um para as linhas da primeira matriz, outro para as colunas da segunda matriz, e um terceiro para somar os produtos dos elementos correspondentes.

A habilidade de utilizar vetores e matrizes em conjunto com loops aninhados é essencial para uma programação eficiente e eficaz. Essa técnica permite a resolução de uma gama de problemas em diversos campos, desde o processamento de imagens e jogos até a análise de dados e cálculos científicos. A compreensão e a aplicação correta dessas ferramentas não apenas melhoram a capacidade de resolver problemas complexos, mas também aumentam a eficiência e a clareza do código, tornando-o mais robusto e fácil de manter.

Dica

A prática contínua e a exploração de diferentes contextos de aplicação são fundamentais para dominar o uso de vetores, matrizes e loops aninhados na programação.

Vetores, matrizes e loops

Descubra neste vídeo como geralmente é abordado o uso de vetores e matrizes com loops. Veja exemplos de como essa estrutura é geralmente feita e seus padrões de escrita mais recomendados.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vetores e matrizes são estruturas de dados fundamentais na programação. Quando combinadas com estruturas de repetição, como loops, elas se tornam recursos indispensáveis para manipulação eficiente de dados. Aqui, vamos explorar como utilizar loops para trabalhar com vetores e matrizes, apresentando exemplos claros e práticos para facilitar a compreensão.

Vetores com estruturas de repetição

Um vetor é uma coleção ordenada de elementos do mesmo tipo. Para manipular os elementos de um vetor, as estruturas de repetição são extremamente úteis. Veja um exemplo básico de como inicializar e exibir os elementos de um vetor.

> Conteúdo interativo
>
> esse a versão digital para executar o código.

Nesse exemplo, usamos um loop **for** para inicializar os elementos do **vetor** vetor e outro loop for para exibir os valores armazenados.

O programa demonstra como utilizar um vetor para armazenar múltiplos valores inteiros e como a estrutura de repetição for pode ser empregada de maneira eficiente tanto para inicializar os valores do vetor quanto para exibi-los.

No primeiro for, o vetor é preenchido com múltiplos de 2, começando de 0 até 8. O segundo for imprime na tela os valores de cada elemento do vetor, neste caso variando de 0 até 8.

Matrizes com estruturas de repetição

Uma matriz é uma coleção bidimensional (ou multidimensional) de elementos do mesmo tipo. As estruturas de repetição são essenciais para percorrer e manipular matrizes. Veja um exemplo de como inicializar e exibir os elementos de uma matriz.

> Conteúdo interativo
>
> esse a versão digital para executar o código.

Aqui, utilizamos dois loops **for** aninhados: o primeiro for começa a varredura pela primeira linha, e a cada passo do loop, a segunda estrutura for é ativada, iniciando a varredura das colunas dessa linha.

Dentro deste segundo for, um simples cálculo é feito: a soma do número da linha com o número da coluna (i + j). Esse valor é atribuído à posição correspondente na matriz, como se você estivesse anotando a soma de duas coordenadas em cada célula do tabuleiro.

Assim, ao final dessa dupla varredura, toda a matriz estará preenchida. Cada elemento terá um valor que corresponde à soma de seu índice de linha com seu índice de coluna. Por exemplo, na posição [0][0], a soma será 0 + 0, resultando em 0. Já na posição [1][2], a soma será 1 + 2, resultando em 3.

Mas preencher a matriz é apenas metade da tarefa. Agora, você precisa exibir esses valores para confirmar que a tabela foi preenchida corretamente. Para isso, você recorre novamente às estruturas for aninhadas.

Desta vez, em vez de atribuir valores, você acessa cada elemento da matriz e o imprime na tela. O programa percorre novamente cada linha e coluna, exibindo os números que foram calculados anteriormente.

Operações combinadas

Podemos realizar operações mais complexas com vetores e matrizes usando estruturas de repetição. Por exemplo, podemos somar duas matrizes ou calcular a soma dos elementos de um vetor.

Soma de matrizes

É uma operação fundamental em programação, especialmente em álgebra linear e computação científica. Consiste em somar os elementos correspondentes de duas matrizes para formar uma nova matriz. Aqui está um exemplo para ilustrar o processo: Vamos somar duas matrizes 2×2. Primeiro, declaramos e inicializamos duas matrizes chamadas matriz1 e matriz2. Em seguida, criamos uma matriz chamada matrizSoma para armazenar os resultados.

Conteúdo interativo

esse a versão digital para executar o código.

Vamos entender o que o código faz:

1. **Declaração e inicialização**: As matrizes matriz1 e matriz2 são declaradas e inicializadas com valores.
2. **Soma das matrizes**: Usamos dois loops for aninhados para percorrer cada elemento das matrizes. Para cada posição [i][j], somamos os valores correspondentes de matriz1 e matriz2, armazenando o resultado em matrizSoma.
3. **Exibição da matriz resultante**: Por fim, usamos outro conjunto de loops for aninhado para percorrer matrizSoma e imprimir seus elementos, mostrando a matriz resultante da soma.

Soma dos elementos de um vetor

É uma operação simples e comum, utilizada para calcular a soma total de todos os elementos presentes em um vetor.

Primeiramente, declaramos e inicializamos um vetor chamado vetor com cinco elementos. Criamos uma variável soma para armazenar o total. Este exemplo mostra como percorrer um vetor e somar seus elementos. Veja!

Conteúdo interativo

esse a versão digital para executar o código.

Vamos entender o que o código faz!

1. **Declaração e inicialização**: O vetor chamado vetor é declarado e inicializado com cinco valores. A variável soma é inicializada com zero.
2. **Soma dos elementos**: Usamos um loop for para percorrer cada elemento do vetor. Em cada iteração, o valor do elemento atual é adicionado à variável soma.
3. **Exibição do resultado**: Por fim, usamos printf para imprimir a soma total dos elementos do vetor.

Vetores e matrizes, quando usados em conjunto com estruturas de repetição, permitem realizar diversas operações de forma eficiente e clara. Eles são fundamentais para a manipulação de coleções de dados em muitas aplicações práticas.

Os exemplos fornecidos demonstram como essas ferramentas podem ser usadas para inicializar, manipular e exibir dados, facilitando a compreensão e a aplicação desses conceitos em problemas reais.

Hora de codar

Neste vídeo, exploraremos o uso de vetores e matrizes com loops, apresentando exemplos práticos para manipulá-los eficientemente. Forneceremos dicas para escrever código claro e otimizado. Este conteúdo é ideal para quem deseja melhorar suas habilidades em manipulação de dados e operações matriciais. Não deixe de conferir!

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

Desafio: nível aventureiro

Tabuleiro Completo e Navios Diagonais

Nesta etapa, você irá aprimorar o seu jogo de Batalha Naval adicionando a complexidade de navios posicionados na diagonal. Dê continuidade ao código que você desenvolveu no Desafio do Nível Novato.

O que você vai fazer

Você deve modificar o seu programa em C para:

1. **Criar um Tabuleiro 10×10:** Declare uma matriz (array bidimensional) de tamanho 10×10 para representar o tabuleiro do Batalha Naval. Inicialize todas as posições com o valor 0, representando água.
2. **Posicionar Quatro Navios:** Posicione quatro navios no tabuleiro.
    - Dois navios devem estar posicionados horizontalmente ou verticalmente (como no nível anterior).
    - Os outros dois navios devem ser posicionados na diagonal. Considere que um navio diagonal ocupa posições onde a linha e a coluna aumentam ou diminuem simultaneamente (ex: tabuleiro[i][i] ou tabuleiro[i][9-i] para um tabuleiro 10×10).
    - Represente as posições ocupadas pelos navios com o valor 3.
    - Escolha as coordenadas iniciais.
    - Valide que as posições dos navios estejam dentro dos limites do tabuleiro e que eles não se sobreponham.
3. **Exibir o Tabuleiro:** Utilize loops aninhados e o comando printf para exibir o tabuleiro completo no console. A saída deve mostrar a matriz 10×10, com 0s representando água e 3s representando as partes dos navios. Utilize espaços para alinhar a saída e facilitar a visualização do tabuleiro.

Requisitos funcionais

- O programa deve utilizar uma matriz 10×10 para representar o tabuleiro.
- O programa deve permitir o posicionamento de quatro navios, sendo dois na diagonal.

- O programa deve validar se as coordenadas e o tamanho dos navios são válidos dentro dos limites do tabuleiro.
- O programa deve garantir que os navios não se sobreponham.
- O programa deve exibir o tabuleiro completo no console com os navios posicionados corretamente.

Requisitos não funcionais

- **Performance:** O programa deve executar de forma eficiente, sem causar atrasos perceptíveis.
- **Documentação:** O código deve ser bem documentado, com comentários claros explicando a lógica e o propósito de cada parte do programa. Utilize comentários para explicar a função de cada variável, loop e bloco de código.
- **Legibilidade:** O código deve ser escrito de forma clara, organizada e fácil de entender, com nomes de variáveis descritivos e indentação consistente.

Simplificações para o nível intermediário

- As coordenadas e tamanhos dos navios são definidos diretamente no código, sem entrada do usuário.
- Os navios têm tamanho fixo igual a 3.
- A validação de sobreposição de navios pode ser simplificada, mas deve cobrir os casos diagonais.
- Não é necessário implementar a lógica do jogo (ataques, acertos, etc.) neste nível.

Entregando seu Projeto

1. **GitHub:** Dê um *push* das suas alterações para o seu repositório no GitHub.
2. **SAVA:** Envie o link do seu repositório GitHub na plataforma SAVA, na atividade correspondente ao Desafio Nível Intermediário. Certifique-se de que o repositório seja público ou que o professor tenha acesso a ele. O link do repositório deve ser **a única forma de entrega do projeto**, garantindo que o professor possa acessar seu código para avaliação.

Lembre-se: Este desafio visa fortalecer suas habilidades em manipulação de matrizes e loops. Concentre-se em produzir um código bem estruturado e comentado, demonstrando sua compreensão dos conceitos. A visualização clara do tabuleiro é essencial para verificar se o seu posicionamento dos navios está correto!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução à manipulação de matrizes com condicionais

Confira neste vídeo as vantagens de utilizar vetores e matrizes com condicionais e loops. Além dos benefícios, explicaremos as dificuldades que geralmente surgem ao combinar essas estruturas.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Importância para programadores

Para programadores, especialmente aqueles que trabalham em áreas como processamento de imagem, análise de dados, simulações científicas e engenharia, a manipulação de matrizes é uma tarefa comum. Conhecer e saber aplicar condicionais em iterações sobre matrizes é importante pelos seguintes motivos que apresentaremos a seguir. Confira!

1

Eficiência e otimização

Condicionais usadas corretamente podem melhorar significativamente a eficiência do seu código. Por exemplo, ao inicializar uma matriz com valores baseados em condições, você pode evitar operações desnecessárias e economizar tempo de execução.

2

Flexibilidade

Condicionais proporcionam uma flexibilidade que permite ao programador aplicar diferentes lógicas de negócio em diferentes partes da matriz. Isso é essencial em aplicações que exigem manipulações específicas de dados.

3

Correção de erros

A habilidade de aplicar condicionais durante a manipulação de matrizes ajuda a garantir a correção do código. Verificações condicionais podem ser usadas para evitar erros comuns, como acessar índices fora dos limites da matriz.

4

Complexidade reduzida

Embora o uso de loops aninhados possa parecer complicado inicialmente, ele simplifica a manipulação de dados em matrizes. Uma vez dominada, essa técnica permite resolver problemas complexos de forma estruturada e compreensível.

Possíveis desafios

Embora a manipulação de matrizes com condicionais seja uma técnica valiosa, ela apresenta alguns obstáculos. Confira quais!

Compreensão e implementação de loops aninhados

Para iniciantes, entender e implementar loops aninhados pode ser intimidante. A complexidade aumenta com o número de dimensões da matriz, exigindo uma compreensão clara da lógica de iteração.

Gestão de índices

A manutenção do controle dos índices durante iterações é crítico. Erros como acessar índices fora dos limites da matriz podem causar falhas no programa e são difíceis de depurar.

Complexidade computacional

Operações condicionais dentro de loops aninhados podem aumentar a complexidade computacional do algoritmo. É essencial considerar a eficiência do código, especialmente ao trabalhar com grandes matrizes.

Legibilidade do código

Com loops aninhados e múltiplas condições, o código pode se tornar difícil de ler. É importante seguir boas práticas de programação, como usar nomes de variáveis descritivos e comentar o código para melhorar a legibilidade.

Depuração e testes

A identificação e a correção de erros em loops aninhados e condicionais podem ser desafiadoras. Ferramentas de depuração e testes abrangentes são essenciais para garantir que o código funcione conforme esperado.

O domínio da manipulação de matrizes com condicionais dentro de loops aninhados é uma habilidade essencial para programadores que trabalham com dados estruturados em múltiplas dimensões. Essa técnica melhora a eficiência e flexibilidade do código e permite a criação de algoritmos confiáveis que podem lidar com operações complexas em matrizes.

Apesar dos desafios, a prática constante e a aplicação de boas práticas de programação podem ajudar a superar as dificuldades, tornando essa habilidade um recurso indispensável no arsenal de qualquer programador.

Condicionais com matrizes e loops

Neste vídeo, abordaremos o uso de condicionais com matrizes e loops, com diversos exemplos explicados e comentados. Através dessa abordagem, vamos entender como realizar operações por meio de acesso aos índices das matrizes. Assista!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O uso de condicionais em conjunto com loops aninhados para manipulação de matrizes é uma técnica fundamental na programação. Através dessa abordagem, podemos inicializar, modificar, contar, substituir e buscar elementos dentro de matrizes, permitindo a implementação de algoritmos eficientes e versáteis.

Inicialização condicional de uma matriz

Inicializar uma matriz com valores baseados em condições específicas é uma tarefa comum. No primeiro exemplo, criamos uma matriz 3×3 em que os elementos são definidos como 1 ou 0, dependendo se o índice da coluna é par ou ímpar. Confira o exemplo!

Conteúdo interativo

esse a versão digital para executar o código.

Explicação do código

Nesse exemplo, queremos criar uma matriz 3×3 e inicializar seus elementos de acordo com a paridade do índice da coluna. Entenda melhor a seguir!

Declaração da matriz

int matriz[3][3]; declara uma matriz de inteiros com 3 linhas e 3 colunas.

Inicialização da matriz

- Utilizamos dois loops for aninhados para percorrer todas as posições da matriz. O loop externo (for (int $i = 0$; $ilt;3$; $i++$)) itera sobre as linhas da matriz, enquanto o loop interno (for (int $j = 0;jlt;3$; $j++$)) itera sobre as colunas.
- Dentro do loop interno, usamos uma condicional if ($j\%2 = 0$) para verificar se o índice da coluna é par. Se a condição for verdadeira, atribuímos o valor 1 ao elemento atual da matriz (matriz[$[1[j] = 1$;). Caso contrário, atribuímos o valor 0 (matriz[$[i[j] = 0$;).

Impressão da matriz

- Após inicializar todos os elementos da matriz, utilizamos outro par de loops for aninhados para percorrer a matriz novamente e imprimir seus valores.
- No loop de impressão, para cada linha, iteramos sobre suas colunas e utilizamos printf("\%d ", matriz[i] [j]); para exibir o valor do elemento atual, seguido de um espaço.
- Após imprimir todos os elementos de uma linha, utilizamos printf("\n"); para pular para a próxima linha, garantindo que a matriz seja exibida de forma organizada.

Modificação condicional de uma matriz

Modificar elementos de uma matriz com base em condições específicas é útil; por exemplo, transformar em negativos os elementos maiores que 5. Confira!

Conteúdo interativo

esse a versão digital para executar o código.

Explicação do código

Nesse exemplo, modificamos uma matriz de forma que todos os elementos maiores que 5 sejam transformados em seus valores negativos. Confira para entender!

Declaração e inicialização da matriz

- int matriz[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}; declara e inicializa uma matriz de inteiros com 3 linhas e 3 colunas.

Modificação condicional dos elementos

- Utilizamos dois loops for aninhados para percorrer as posições da matriz. O loop externo (for (int i = 0; i < 3; i++)) itera sobre as linhas da matriz, enquanto o loop interno (for (int $j = 0; jlt; 3; j++$)) itera sobre as colunas.
- Dentro do loop interno, usamos uma condicional if (matriz[][] > 5) para verificar se o valor do elemento atual é maior que 5. Se a condição for verdadeira, transformamos o valor do elemento em seu valor negativo (matriz[][] = -matriz[i][[];).

Contagem condicional em uma matriz

Contar elementos que atendem a determinadas condições dentro de uma matriz é importante; no terceiro exemplo, contamos quantos elementos são pares e quantos são ímpares.

Contagem condicional em uma matriz

Contar elementos que atendem a determinadas condições dentro de uma matriz é importante; no terceiro exemplo, contamos quantos elementos são pares e quantos são ímpares.

Conteúdo interativo

esse a versão digital para executar o código.

Explicação do código

Nesse exemplo, queremos contar quantos elementos são pares e quantos são ímpares. Confira para entender!

Declaração e inicialização da matriz

- int matriz[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}; declara e inicializa uma matriz de inteiros com 3 linhas e 3 colunas.

Declaração de contadores

- int evenCount $= 0$, oddCount $= 0$; declara e inicializa dois contadores, um para os elementos pares (evenCount) e outro para os impares (oddCount), ambos iniciados em 0.

Contagem dos elementos

- Utilizamos dois loops for aninhados para percorrer todas as posições da matriz. O loop externo (for (int $i = 0$; $ilt;3$; $i++$)) itera sobre as linhas da matriz, enquanto o loop interno (for (int $j = 0; jlt;3$; $j++$)) itera sobre as colunas.
- Dentro do loop interno, usamos uma condicional if (matriz[i][] \% $2 = 0$) para verificar se o valor do elemento atual é par. Se a condição for verdadeira, incrementamos o contador de elementos pares (evenCount++). Caso contrário, incrementamos o contador de elementos ímpares (oddCount++).

Impressão dos resultados

- Após percorrer toda a matriz e contar os elementos pares e ímpares, utilizamos printf para imprimir os resultados.
- printf("Número de elementos pares: %d\n", evenCount); imprime o número total de elementos pares.
- printf("Número de elementos ímpares: %d\n", oddCount); imprime o número total de elementos ímpares.

Substituição condicional de elementos

Substituir elementos de uma matriz com base em condições específicas é útil; no quarto exemplo, os elementos múltiplos de 3 são substituídos por -1. Veja o exemplo!

Conteúdo interativo

esse a versão digital para executar o código.

Explicação do código

Nesse exemplo, modificamos uma matriz de forma que todos os elementos que são múltiplos de 3 sejam substituídos por -1. Confira para entender!

Declaração e inicialização da matriz

- A matriz matriz é declarada e inicializada com valores predefinidos: $\{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}$.

Substituição condicional dos elementos

- Utilizamos dois loops for aninhados para percorrer todas as posições da matriz. O loop externo (for (int $i=0;ilt;3$; $i++$)) itera sobre as linhas da matriz, enquanto o loop interno (for (int $j=0;jlt;3$; $j++$)) itera sobre as colunas.
- Dentro do loop interno, usamos uma condicional if (matriz[][]] \% $3=0$) para verificar se o valor do elemento atual é múltiplo de 3 . Se a condição for verdadeira, substituímos o valor do elemento por - -1 (matriz[ ][] $=-1$).

Impressão da matriz modificada

- Após modificar os elementos da matriz, utilizamos outro par de loops for aninhados para percorrer a matriz novamente e imprimir seus valores.
- No loop de impressão, para cada linha, iteramos sobre suas colunas e utilizamos printf("%d ", matriz[j][j]) para exibir o valor do elemento atual, seguido de um espaço.
- Após imprimir todos os elementos de uma linha, utilizamos printf("\n") para pular para a próxima linha, garantindo que a matriz seja exibida de forma organizada.

Busca condicional em uma matriz

A busca condicional é essencial para localizar elementos específicos em uma matriz; no quinto exemplo, procuramos um valor específico e imprimimos sua posição, se encontrado. Confira!

Conteúdo interativo

esse a versão digital para executar o código.

Explicação do código

Nesse exemplo, buscamos um valor específico em uma matriz e imprimir sua posição, se encontrado. Confira!

Declaração e inicialização da matriz

- A matriz matriz é declarada e inicializada com valores predefinidos: $\{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}$.

Declaração do valor alvo

- Declaramos um valor alvo (target) para procurar na matriz: int target = 5;
- Declaramos uma variável de controle (found) para indicar se o valor foi encontrado: int found = 0;

Busca condicional do elemento alvo

- Utilizamos dois loops for aninhados para percorrer todas as posições da matriz. O loop externo (for (int $i = 0; ilt; 3$; $i++$)) itera sobre as linhas da matriz, enquanto o loop interno (for (int $j = 0; jlt; 3$; $j++$) itera sobre as colunas.
- Dentro do loop interno, usamos uma condicional if (matriz[][] == target) para verificar se o valor do elemento atual é igual ao valor alvo. Se a condição for verdadeira, imprimimos a posição do elemento na matriz printf("Elemento %d encontrado na posição (%d, %d)\n", target, i, j);), definimos a variável found como 1 para indicar que o valor foi encontrado e utilizamos break para sair do loop interno.
- Após o break do loop interno, utilizamos outro if (found) para sair do loop externo, interrompendo a busca assim que o valor é encontrado.

Verificação e impressão do resultado

- Após percorrer a matriz, verificamos se o valor alvo foi encontrado. Se found ainda for 0 , significa que o valor não foi encontrado na matriz. Utilizamos printf("Elemento %d não encontrado na matriz\n", target); para imprimir uma mensagem indicando que o valor alvo não foi encontrado.

Hora de codar

Explore neste vídeo o uso de condicionais com matrizes e loops aninhados. Aprenda a aplicar condições para manipular matrizes eficientemente, com exemplos práticos e dicas para inicializar, modificar, contar, substituir e buscar elementos, melhorando suas habilidades em dados multidimensionais.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível mestre

Habilidades especiais e áreas de efeito

Neste desafio final, você adicionará um toque estratégico ao seu jogo de Batalha Naval, implementando habilidades especiais com áreas de efeito distintas. Você continuará trabalhando no mesmo projeto iniciado nos níveis anteriores, adicionando a lógica para representar e exibir essas habilidades no tabuleiro.

O que você vai fazer

Você deve modificar o seu programa em C para:

1. **Definir o Tabuleiro:** Mantenha o tabuleiro 10×10 que você criou no nível anterior. Ele servirá como base para visualizar as áreas de efeito das habilidades.

2. **Criar Matrizes de Habilidade:** Crie três matrizes separadas, cada uma representando a área de efeito de uma habilidade especial:

    - **Cone:** Uma matriz que represente uma área em forma de cone, com o ponto de origem no topo e expandindo-se em direção à base. Imagine um cone apontando para baixo. A matriz deve representar essa forma, com o ponto de origem (o topo do cone) na parte superior e a área se expandindo para baixo.

    - **Cruz:** Uma matriz que represente uma área em forma de cruz, com o ponto de origem no centro.

    - **Octaedro:** Uma matriz que represente a vista frontal de um octaedro (imagine como se estivesse olhando diretamente para ele), resultando em um formato que se assemelha a um losango, com o ponto de origem no centro.

    - As matrizes de habilidade devem indicar quais posições são afetadas pela habilidade com o valor 1, e quais não são afetadas com o valor 0. Defina um tamanho razoável para as matrizes de habilidade (por exemplo, 5×5 ou 7×7).

3. **Integrar Habilidades ao Tabuleiro**:

    - Defina um ponto de origem para cada habilidade no tabuleiro (coordenadas linha e coluna). Este ponto de origem será o centro da habilidade, a partir do qual a área de efeito será calculada.

    - Crie a lógica para sobrepor a matriz de habilidade ao tabuleiro, centrando a matriz de habilidade no ponto de origem definido.

    - As posições do tabuleiro que estiverem dentro da área de efeito da habilidade (valor 1 na matriz de habilidade) devem ser marcadas visualmente. Utilize um valor diferente de 0 e 3 para representar as posições afetadas pela habilidade (por exemplo, o valor 5).

4. **Exibir o Tabuleiro com Habilidade:** Utilize loops aninhados e o comando printf para exibir o tabuleiro no console, mostrando as áreas afetadas pelas habilidades. Utilize caracteres diferentes para representar:

    - Água (0)

    - Navio (3)

    - Área afetada pela habilidade (5)

5. **Utilizar Condicionais:** As matrizes de habilidades devem ser construídas de forma dinâmica, utilizando condicionais dentro de loops aninhados. A sobreposição das habilidades no tabuleiro também deve utilizar condicionais para garantir que a área de efeito permaneça dentro dos limites do tabuleiro.

Exemplo de saída de habilidade em cone:

0 0 3 0 0

0 3 3 3 0

3 3 3 3 3

Exemplo de saída de habilidade em cruz:

0 0 3 0 0

3 3 3 3 3

0 0 3 0 0

Exemplo de saída de habilidade em octaedro:

0 0 3 0 0

0 3 3 3 0

0 0 3 0 0

Requisitos funcionais

- O programa deve criar as matrizes de efeito para as habilidades Cone, Cruz e Octaedro.
- As matrizes de habilidade devem utilizar os valores 0 e 1 para indicar as áreas não afetadas e afetadas, respectivamente.
- O programa deve permitir definir um ponto de origem para cada habilidade no tabuleiro.
- O programa deve sobrepor as matrizes de habilidade ao tabuleiro, marcando as áreas afetadas.
- A saída do programa deve exibir o tabuleiro com os navios e as áreas de efeito das habilidades, utilizando diferentes caracteres para cada elemento.
- A construção das matrizes de habilidade e a sobreposição no tabuleiro devem obrigatoriamente utilizar estruturas de repetição aninhadas e condicionais.

Requisitos não funcionais

- **Performance:** O programa deve executar de forma eficiente, sem causar atrasos perceptíveis.
- **Documentação:** O código deve ser bem documentado, com comentários claros explicando a lógica e o propósito de cada parte do programa. Utilize comentários para explicar como as áreas de efeito das habilidades são calculadas.
- **Legibilidade:** O código deve ser escrito de forma clara, organizada e fácil de entender, com nomes de variáveis descritivos e indentação consistente.

Simplificações para o nível avançado

- A posição das habilidades no tabuleiro é definida diretamente no código, sem input do usuário.
- Não é necessário implementar a lógica de dano ou efeito das habilidades nos navios (apenas a visualização da área de efeito).

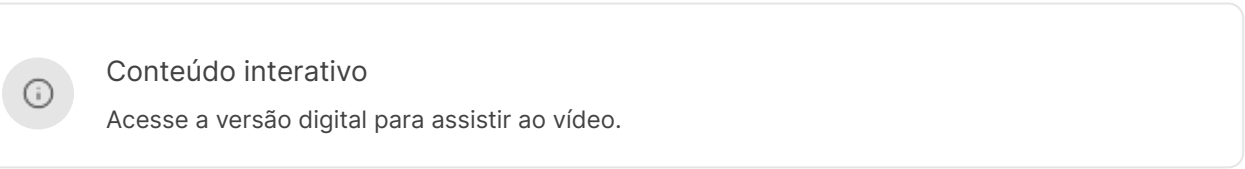
- O tamanho das matrizes de habilidade pode ser fixo.
- A validação de limites do tabuleiro durante a sobreposição da habilidade pode ser simplificada.

Entregando seu Projeto

1. **GitHub:** Dê um *push* das suas alterações para o seu repositório no GitHub.
2. **SAVA:** Envie o link do seu repositório GitHub na plataforma SAVA, na atividade correspondente ao Desafio Nível Avançado. Certifique-se de que o repositório seja público ou que o professor tenha acesso a ele. O link do repositório deve ser **a única forma de entrega do projeto**, garantindo que o professor possa acessar seu código para avaliação.

Este desafio é a culminação do seu aprendizado sobre vetores e matrizes. Demonstre sua capacidade de manipular essas estruturas de dados de forma criativa e eficiente, integrando as habilidades especiais ao tabuleiro do jogo. Um código bem documentado e uma exibição clara das matrizes de habilidade sobrepostas ao tabuleiro são essenciais para o sucesso neste desafio!

Nos módulos seguintes, deverá haver orientação de atualização do repositório github com o incremento desenvolvido e sugestão de mensagem para o commit.



Considerações finais

Você concluiu o aprendizado sobre matrizes e vetores! Esse é um marco importante na sua jornada como programador, pois agora você domina um dos conceitos fundamentais para a manipulação de dados. Compreender e aplicar vetores e matrizes abre diversas possibilidades, desde a ciência de dados até o desenvolvimento de algoritmos eficientes.

Sua dedicação e esforço para entender esses conceitos complexos são admiráveis. Agora, você está preparado para enfrentar desafios mais avançados, aplicar seu conhecimento em projetos práticos e otimizar seu código com mais eficácia. Lembre-se de que a prática contínua é essencial para solidificar seu aprendizado.

Continue explorando, praticando e aprimorando suas habilidades. Seu compromisso com o crescimento e a excelência é inspirador. Que esse sucesso seja apenas o começo de muitas conquistas futuras na sua trajetória de programação!

Referências

W3SCHOOLS. **C Tutorial.** Consultado na internet em: 15 jul. 2024.

ISO/IEC. **Programming Languages - C**. Consultado na internet em: 15 jul. 2024.

PUC-RS. **Matrizes e Vetores**. Porto Alegre: PUC-RS. Consultado na internet em: 15 jul. 2024.

MICROSOFT. **Copilot**. Consultado na internet em: 15 jul. 2024.