Super Trunfo em C: desenvolvendo a lógica do jogo

Você irá desenvolver um programa em C para elaborar uma aplicação que simula o jogo Super Trunfo, empregando estruturas de decisão.

Curadoria de TI

1. Itens iniciais

Objetivos

- Desenvolver um programa que compare dois atributos de cartas do Super Trunfo utilizando estruturas de decisão.
- Desenvolver o programa com menus interativos e implementar comparações de cartas usando estruturas de decisão aninhadas e encadeadas.
- Desenvolver o programa com funcionalidades avançadas, como a comparação de dois atributos e emprego de operadores ternários.

Introdução

Bem-vindo ao desafio de programação onde você construirá um jogo Super Trunfo em C! Imagine que você foi contratado pela TechNova, uma empresa inovadora de desenvolvimento de jogos, para aprimorar a versão digital do clássico Super Trunfo. Seu objetivo é criar um jogo dinâmico e envolvente, utilizando estruturas de decisão para implementar a lógica de comparação entre as cartas e menus interativos para aprimorar a experiência do jogador.

No mercado de trabalho da área de Tecnologia da Informação, a capacidade de programar a lógica de um software, como jogos, utilizando estruturas de decisão é fundamental. Através delas, os programas ganham a capacidade de reagir de forma diferente a diferentes situações, tornando-se mais interativos e inteligentes. Dominar esse conhecimento é essencial para construir aplicações robustas e eficientes, e este desafio prático te colocará à frente nesse processo.

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

Cenário

A TechNova, empresa de desenvolvimento de jogos, decidiu modernizar o clássico Super Trunfo. A versão digital do jogo precisa de uma lógica de comparação de cartas mais robusta e de menus interativos que permitam ao jogador escolher os atributos para a comparação, tornando o jogo mais dinâmico e envolvente. A empresa te contratou para implementar essas melhorias utilizando a linguagem C e seus conhecimentos em estruturas de decisão.

Sua Missão?

Você deverá desenvolver um programa em C que simule o jogo Super Trunfo. O programa deverá:

- Implementar a lógica de comparação entre duas cartas, considerando diferentes atributos numéricos.
- Permitir ao jogador escolher entre diferentes atributos para a comparação através de menus interativos.

- Evoluir em complexidade ao longo dos três desafios:
    - Comparação de cartas com base em um único atributo utilizando if e if-else.
    - Comparação de cartas com múltiplos atributos usando operadores lógicos e estruturas de decisão aninhadas e encadeadas (if-else if), além da implementação de menus com switch.
    - Comparação de cartas com base em dois atributos, utilizar o operador ternário e integrando todas as estruturas de decisão aprendidas para criar uma lógica mais complexa.

Introdução às estruturas de decisão

Aprenda neste vídeo como utilizar estruturas de decisão e operadores relacionais para criar programas mais inteligentes. Além disso, descubra a importância dessas ferramentas com exemplos práticos em pseudocódigo, ajudando você a tomar decisões no seu código.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A importância das estruturas de decisão

Primeiramente, vamos falar sobre as estruturas de decisão, que são fundamentais para qualquer linguagem de programação.

Imagine que você esteja em um cruzamento e precisa decidir se vai virar à direita, à esquerda ou seguir em frente. Essa decisão depende das condições ao seu redor, como sinais de trânsito ou para onde você deseja ir.

> No mundo da programação, estruturas de decisão fazem algo semelhante: elas permitem que seu programa tome decisões com base em certas condições.

Sem as estruturas de decisão, nossos programas seriam como trens em trilhos fixos, sempre seguindo a mesma rota sem considerar as mudanças no ambiente. Com as estruturas de decisão, nossos programas podem reagir de maneiras diferentes a várias entradas, tornando-os muito mais flexíveis e poderosos.

Operadores relacionais em estruturas de decisão

Para tomar decisões, precisamos comparar valores. É aí que entram os operadores relacionais. Eles comparam dois valores e retornam verdadeiro (true) ou falso (false). Aqui estão os operadores relacionais que você usará com mais frequência:

- **>** : Maior que
- **<** : Menor que
- **>=** : Maior ou igual a
- **<=** : Menor ou igual a
- **==** : Igual a
- **!=** : Diferente de

Observação: Além dos operadores relacionais, também existem os operadores lógicos que podem ser empregados para realizar a comparação.

Os operadores relacionais, em conjunto com os operadores lógicos, são empregados nas estruturas de decisão para determinar o caminho a ser seguido na lógica do programa, ao comparar duas variáveis e verificar qual é maior, menor, igual ou diferente. Assim, uma condição será testada para verificar se é verdadeira ou falsa. Dependendo do resultado da condição, notamos o seguinte. Veja!

Se a condição for verdadeira

Seguiremos por um caminho do programa.

Se a condição for falsa

Seguiremos por outro caminho.

Se você quiser saber se a quantidade de maçãs é maior que a quantidade de laranjas, você usaria o operador >. Acompanhe um exemplo em pseudocódigo.

```
SE laranja > maçã ENTÃO
ESCREVA "Tem mais laranja que maçã"
SENÃO
ESCREVA "Tem menos laranja que maçã"
FIM_SE
```

Neste exemplo, o programa compara os valores de laranja e maçã. Se houver mais laranja que maçã, ele escreve "Tem mais laranja que maçã". Caso contrário, ele escreve " Tem menos laranja que maçã".

Exemplos práticos de utilização

Verificação de idade para votação

Imagine criar um programa que verifica se uma pessoa tem idade suficiente para votar. No Brasil, a idade mínima para votar é 16 anos. Veja como fazer isso em pseudocódigo:

```
SE IDADE >= 16 ENTÃO
ESCREVA "Você pode votar!"
SENÃO
ESCREVA "Você ainda não pode votar."
FIM_SE
```

Aqui, estamos usando o operador >= para verificar se a idade é maior ou igual a 16. Se for, o programa escreve "Você pode votar!". Caso contrário, escreve "Você ainda não pode votar.".

Verificação de nota

Imagine um programa que precisa verificar se um estudante passou em um exame. Vamos supor que a nota mínima para passar seja 60.

```
SE NOTA >= 60 ENTÃO
ESCREVA "Parabéns, você passou!"
SENÃO
ESCREVA "Infelizmente, você não passou."
FIM_SE
```

Nesse caso, o operador >= é usado novamente para verificar se a nota é maior ou igual a 60. Dependendo do resultado, o programa escreve uma mensagem apropriada.

Comparação de preços

Vamos criar um programa que compara os preços de dois produtos e informa qual é mais barato.

```
PRECO1 ← 50.0
PRECO2 ← 75.0
SE PRECO1 < PRECO2 ENTÃO
ESCREVA "O produto 1 é mais barato."
SENÃO_SE PRECO1> PRECO2 ENTÃO
ESCREVA "O produto 2 é mais barato."
SENÃO
ESCREVA "Os produtos têm o mesmo preço."
FIM_SE
```

Aqui, usamos os operadores < e > para comparar os preços. Se PRECO1 for menor que PRECO2, o programa escreve "O produto 1 é mais barato". Se PRECO1 for maior, ele escreve "O produto 2 é mais barato". Se os preços forem iguais, ele escreve "Os produtos têm o mesmo preço".

Considerando que PRECO1 recebe o valor 50 e PRECO2 recebe o valor 75, seria impresso “O produto 1 é mais barato.”.

Agora que você entendeu a importância das estruturas de decisão e como usar operadores relacionais, você está pronto para criar programas mais inteligentes e dinâmicos. Lembre-se, a prática é essencial. Experimente criar seus próprios exemplos e veja como diferentes condições podem alterar o comportamento do seu programa.

Ainda vamos explorar mais estruturas de decisão para aumentar a complexidade e a funcionalidade dos seus programas. Boa codificação!

Estruturas de decisão simples

Aprenda neste vídeo a utilizar a estrutura if em C para criar programas dinâmicos e inteligentes. Veja a definição, aplicação e os exemplos práticos de comparação de valores, além de exercícios para fixação. Conheça as estruturas de decisão e controle o fluxo do seu código de forma eficiente.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Definição e aplicação do if

Agora que você já compreendeu a importância das estruturas de decisão, vamos explorar uma das mais básicas e essenciais: a estrutura **if**. Essa estrutura permite que um programa execute um bloco de código

apenas se uma condição específica for verdadeira. Pense nisso como uma simples verificação: "Se isso for verdade, então faça isso".

Vamos iniciar com a definição do comando if em C.

```c
if (condicao) {
  // bloco de código a ser executado
}
```

Aqui, **condicao** é uma expressão que pode ser verdadeira (true) ou falsa (false). Se a condição for verdadeira, o bloco de código dentro do if será executado. Caso contrário, o programa continuará a executar o código que vem depois do bloco if.

A condição a ser testada pode empregar operadores relacionais, lógicos ou usar um número.

Dica

Na linguagem C, o número 0 é considerado falso e qualquer número diferente de 0 é verdadeiro. Portanto, se você colocar o valor 0 no local da condição, o bloco de código nunca será executado. Qualquer outro valor numérico tornará a condição verdadeira e o bloco de código será executado. Este comportamento se assemelha ao tipo booleano, onde 0 representa 'falso' e qualquer valor diferente de 0 representa 'verdadeiro'.

Vamos conferir alguns exemplos!

Comparação de dois valores

Para entender melhor como o if funciona, imagine comparar dois números e mostrar qual deles é maior.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, o programa verifica se numero1 é maior que numero2. Se for, ele exibe a mensagem "numero1 é maior que numero2". Caso contrário, nada será exibido. Nesse exemplo, a condição numero1 > numero2 (8 > 5) é verdadeira. Então, o programa executa o comando dentro do if.

Fique agora com outro exemplo no qual verificamos se uma pessoa é maior de idade.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, estamos verificando se idade é maior ou igual a 18. Se for, como no exemplo, o programa exibe a mensagem "Você é maior de idade".

Verificação de temperatura

O programa verifica se a temperatura está superior a 30 graus e exibe uma mensagem indicando que está calor. Para o valor apresentado, a condição será verdadeira.

Conteúdo interativo

esse a versão digital para executar o código.

Verificação de nota

O programa verifica se a nota de um aluno é maior ou igual a 60 e exibe uma mensagem indicando se o aluno passou. Se a condição for atendida, a mensagem exibida será: "Você Passou!".

Conteúdo interativo

esse a versão digital para executar o código.

Como o valor da nota é 75, o aluno passou.

Comparação de idades

Agora, as idades de duas pessoas são comparadas e é exibida uma mensagem indicando qual delas é mais velha. O que seria impresso neste código?

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, não seria impresso nada, pois a condição é falsa.

Verificação de número par

O programa confere se um número é par e exibe uma mensagem indicando que o número é par. O operador % na linguagem C é conhecido como o operador de módulo ou resto da divisão inteira. Ele retorna o resto de uma divisão entre dois números inteiros, por exemplo a % b. Quando você divide a por b, o operador % retorna o resto dessa divisão. Veja!

Conteúdo interativo

esse a versão digital para executar o código.

Como o resto da divisão de 4 por 2 é 0, será impresso "O número é par".

Verificação de estoque

O exemplo a seguir verifica se a quantidade de um produto está abaixo de um nível crítico (por exemplo, 5 unidades) e exibe uma mensagem indicando que o estoque está baixo. Qual seria a saída deste programa?

Conteúdo interativo

esse a versão digital para executar o código.

A saída desse programa seria: "Estoque baixo".

Assim, as estruturas de decisão simples como o if, são fundamentais para controlar o fluxo do seu programa com base em condições específicas. Elas permitem que você faça verificações e tome decisões dentro do seu código, tornando seus programas mais inteligentes e dinâmicos. Pratique bastante e experimente criar suas próprias condições e verificações. Modifique os programas para que você possa entrar com os valores e testar as condições.

Ainda vamos explorar estruturas de decisão mais complexas para aumentar a capacidade de controle do seu código. Boa codificação!

Estruturas de decisão composta

Aprenda neste vídeo a utilizar a estrutura if-else para criar programas que tomam decisões com base em condições específicas. Além disso, entenda a definição e aplicação do if-else em C com exemplos práticos.

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

Definição e aplicação do if-else

Agora que já entendemos como utilizar a estrutura if, vamos para uma ferramenta um pouco mais avançada: a estrutura **if-else**. Essa estrutura permite ao programa escolher entre dois blocos de código diferentes, dependendo se a condição é verdadeira ou falsa. Pense nisso como uma bifurcação em uma estrada: se a condição for verdadeira, o programa segue por um caminho; se for falsa, ele segue por outro.

Conheça a seguir a sintaxe do if-else em C.

```
c

if (condicao) {
  // bloco de código a ser executado se a condição for verdadeira
} else {
  // bloco de código a ser executado se a condição for falsa
}
```

Agora, confira um exemplo simples para entender melhor.

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

Aqui, o programa verifica se numero é divisível por 2. O operador % retorna o resto da divisão. Se o resto da divisão de numero por 2 for igual a 0, então o número é par e imprime "O número é par". Caso contrário, ele imprime "O número é ímpar".

Comparação de valores

Vamos explorar mais alguns exemplos de comparação de valores utilizando **if-else**.

Verificação de temperatura

O programa verifica se a temperatura está superior a 30 graus e exibe uma mensagem indicando que está calor. Caso contrário, exibe uma mensagem indicando que não está calor.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, a variável temperatura recebe o valor 25. Assim, a saída do programa será “Não está calor”.

Verificação de nota

O programa verifica se a nota de um aluno é maior ou igual a 60 e exibe uma mensagem indicando que o aluno passou. Caso contrário, exibirá uma mensagem indicando que o aluno não passou. Será que o aluno passou? Confira!

Conteúdo interativo

esse a versão digital para executar o código.

Podemos observar que a nota é 65. Portanto, o aluno passou e o programa imprimirá: “Você passou!”.

Verificação de idade

O programa verifica se uma pessoa é maior de idade (18 anos ou mais) e exibe uma mensagem indicando que a pessoa é maior de idade. Caso contrário, exibe uma mensagem indicando que a pessoa é menor de idade. Veja!

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, como a variável idade recebe o valor 16, a saída do programa será: “Você é menor de idade”.

Comparação de preços

Como seria um programa para comparar o preço de um produto e dizer se ele está caro a partir de determinado valor? O programa a seguir compara o preço de um produto com um valor fixo (100.0) e exibe uma mensagem indicando se o produto é caro ou barato. O produto está caro ou barato?

Conteúdo interativo

esse a versão digital para executar o código.

O produto, nesse caso, está caro, pois a variável “**preco**” recebe 120.00, que é maior que 100.

Verificação de estoque

O programa a seguir verifica se o estoque de um produto está abaixo de um nível crítico (por exemplo, 5 unidades) e exibe uma mensagem indicando que o estoque está baixo. Caso contrário, exibe uma mensagem indicando que o estoque está adequado.

Conteúdo interativo

esse a versão digital para executar o código.

A saída do programa aqui será “Estoque baixo” porque a variável estoque recebeu o valor 3, que é menor que 5.

Com base nisso, podemos concluir que as estruturas de decisão composta, como o if-else, permitem que seu programa escolha entre dois caminhos diferentes, tornando-o mais flexível e adaptável a diferentes situações. Pratique bastante para se familiarizar com essas estruturas e experimentar diferentes condições e verificações. Modifique os programas para que você possa digitar os valores e testar as condições.

Vamos explorar outras formas de controle de fluxo para ampliar ainda mais as capacidades dos seus programas. Boa codificação!

Hora de codar

Chegou a hora de colocar a mão na massa! No nível novato, aprendemos sobre as estruturas de decisão if e if-else. Praticamos com exemplos como verificar números pares e ímpares, calcular a aprovação de alunos com base em suas notas e comparar preços de produtos.

Neste vídeo, vamos consolidar esse conhecimento com exercícios práticos que mostram como usar if e if-else para criar programas mais dinâmicos e interativos. Você verá como essas estruturas são a base para construir lógicas mais complexas que serão apresentadas nos próximos níveis.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível novato

Comparando Cartas do Super Trunfo

Você já cadastrou as cartas com suas informações no desafio anterior (incluindo estado, nome, população, área, PIB, e pontos turísticos, além da densidade populacional e PIB per capita calculados). Agora, você irá implementar a lógica do jogo!

O que você vai fazer

Você implementará a lógica para comparar duas cartas e determinar a vencedora com base em um único atributo numérico (como população, área, PIB, etc.). Este desafio é uma continuação do desafio do tema anterior, onde você desenvolveu o cadastro das cartas e você poderá reaproveitar o código de cadastro de cartas que já foi desenvolvido. O foco deste nível é a comparação entre duas cartas já cadastradas, utilizando estruturas de decisão if e if-else.

Requisitos funcionais

Seu programa em C deverá:

1. **Receber os dados de duas cartas:** O programa deve receber os dados de duas cartas do Super Trunfo. Utilize o código desenvolvido no desafio anterior para o cadastro das cartas. As cartas devem conter os seguintes atributos:
    - Estado (string)

- Código da carta (string)
- Nome da cidade (string)
- População (int)
- Área (float)
- PIB (float)
- Número de pontos turísticos (int)

2. **Calcular Densidade Populacional e PIB per capita:** O programa deve calcular e exibir:
    - Densidade Populacional: População / Área
    - PIB per capita: PIB / População

3. **Comparar um atributo escolhido:** Você deverá escolher **apenas um** dos atributos numéricos (População, Área, PIB, Densidade Populacional ou PIB per capita) para realizar a comparação entre as duas cartas. **Essa escolha deve ser feita diretamente no código**, não pela entrada do usuário.

4. **Determinar a carta vencedora:**
    - Para todos os atributos, exceto Densidade Populacional, a carta com o **maior valor** vence.
    - Para Densidade Populacional, a carta com o **menor valor** vence.

5. **Exibir o resultado da comparação:** O programa deve exibir, de forma clara, qual carta venceu a comparação, incluindo o atributo utilizado na comparação e os valores das duas cartas para aquele atributo. *Exemplo:*

Comparação de cartas (Atributo: População):

Carta 1 - São Paulo (SP): 12.300.000

Carta 2 - Rio de Janeiro (RJ): 6.000.000

Resultado: Carta 1 (São Paulo) venceu!

Requisitos não funcionais

- **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. Utilize nomes de variáveis descritivos.
- **Documentação:** Comente seu código, explicando o propósito de cada parte.

Simplificações para o nível básico

- Neste nível, você irá comparar apenas **duas cartas** pré-definidas no código.

- A escolha do atributo para comparação será feita **diretamente no código**, não haverá interação com o usuário para escolher o atributo.

- Não é necessário implementar menus interativos neste nível. Foque na lógica de comparação utilizando if e if-else.

Entregando seu Projeto

1. **Desenvolva seu projeto no GitHub:** Continue usando 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:** Compile e teste seu programa rigorosamente, 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:** Envie o link do seu repositório no GitHub através da plataforma SAVA.

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.

Operadores lógicos

Aprenda neste vídeo a utilizar operadores lógicos em C para criar condições complexas. A partir disso, entenda os operadores lógicos &&, ||, !, a precedência de operadores e como combiná-los com operadores relacionais.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução aos operadores lógicos: &&, ||, !

Os operadores lógicos possibilitam a criação de condições complexas e permitem avaliar múltiplas condições de maneira eficiente. Eles são amplamente utilizados em laços de repetição, estruturas condicionais e na verificação de estados em programas de qualquer complexidade.

A partir disso, vamos conhecer os principais operadores lógicos em C.

E Lógico (&&)

Descrição: o operador && retorna verdadeiro se e somente se ambas as expressões que ele conecta forem verdadeiras. Se qualquer uma das expressões for falsa, o resultado será falso.

Sintaxe: expressao1 && expressao2

Exemplo:

```
c

if (a > 0 && b > 0) {
  printf("Ambos são positivos.\n");
}
```

Nesse exemplo, a mensagem será impressa apenas se tanto **a** quanto **b** forem maiores que 0.

OU Lógico (II)

Descrição: o operador || retorna verdadeiro se pelo menos uma das expressões que ele conecta for verdadeira. Se ambas as expressões forem falsas, o resultado será falso.

Sintaxe: expressao1 || expressao2

Exemplo:

```c
if (a > 0 || b > 0) {
  printf("Pelo menos um é positivo.\n");
}
```

Nesse exemplo, a mensagem será impressa se **a** ou **b** (ou ambos) forem maiores que 0.

NÃO Lógico (!)

Descrição: o operador ! inverte o valor lógico de uma expressão. Se a expressão for verdadeira, ele retorna falso, e se a expressão for falsa, ele retorna verdadeiro.

Sintaxe: !expressao

Exemplo:

```c
if (!a) {
  printf("a é zero.\n");
}
```

Neste exemplo, a mensagem será impressa se 'a' for zero, pois a negação de zero (!0) resulta em um valor diferente de zero, que é considerado verdadeiro em C.

Tabela verdade dos operadores lógicos

É uma ferramenta útil para entender como os operadores lógicos funcionam. Ela mostra todas as combinações possíveis de valores verdadeiros (true) e falsos (false) para as condições e os resultados correspondentes dos operadores lógicos.

Dica

É comum referenciar o valor verdadeiro como 1 e o valor falso como 0. Porém, na linguagem C, qualquer valor diferente de 0 é verdadeiro.

Veja a seguir as tabelas verdade para os principais operadores lógicos.

Operador AND Lógico (&&)

Para compreender o funcionamento do operador AND lógico (&&), observe!

A	B	A && B
1	1	1
1	0	0

A	B	A && B
0	1	0
0	0	0

Tabela: Representação do operador &&.
Sérgio dos Santos Cardoso Silva.

Nessa tabela, notamos que o **operador &&** retorna verdadeiro apenas se ambas as condições forem verdadeiras.

Operador OR Lógico (||)

Para entender como o operador OR lógico (||) opera, veja a tabela a seguir.

A	B	A || B
1	1	1
1	0	1
0	1	1
0	0	0

Tabela: Representação do operador ||.
Sérgio dos Santos Cardoso Silva.

Com base nisso, percebemos que o **operador ||** retorna verdadeiro se pelo menos uma das condições for verdadeira.

Operador NOT Lógico (!)

Para conhecer o comportamento do operador NOT lógico (!), confira a próxima tabela.

A	!A
true	false
false	true

Tabela: Representação do operador (!).
Sérgio dos Santos Cardoso Silva.

Assim, percebemos que o **operador !** inverte o valor lógico da condição. Se a condição for verdadeira, ela se torna falsa, e vice-versa.

Exemplos com operadores lógicos

Vamos começar com alguns exemplos simples para entender como esses operadores funcionam.

Operador AND lógico (&&)

Esse operador é utilizado quando queremos que todas as condições sejam verdadeiras para que a expressão inteira seja verdadeira.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, a condição a > 0 && b > 0 é verdadeira porque tanto a quanto b são maiores que zero, ou seja, a > 0 é verdadeiro e b > 0, também é verdadeiro. Se as duas são verdadeiras, o resultado é verdadeiro.

Portanto, o programa irá imprimir “Ambos os números são positivos”.

Operador OR lógico (||)

Esse operador é utilizado quando queremos que pelo menos uma das condições seja verdadeira para que a expressão inteira seja verdadeira.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, a condição $agt;0\|bgt;0$ é verdadeira porque a é maior que zero, mesmo que b não seja. Portanto, o programa irá imprimir "Pelo menos um dos números é positivo".

Operador NOT lógico (!)

Esse operador é utilizado para inverter o valor lógico de uma condição. Se a condição for verdadeira, ela se torna falsa, e vice-versa.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, a condição ! $agt;0)$ é verdadeira porque $agt;0$ é falsa, e o operador ! inverte isso para verdadeiro. 0 resultado do programa será a impressão de "a não é um número positivo".

Precedência de operadores

Determina a ordem em que as operações são realizadas. Em C, a precedência dos operadores lógicos é importante para garantir que as condições sejam avaliadas corretamente.

> Os operadores de maior precedência serão avaliados primeiro, depois vão sendo avaliados os de menor precedência.

Na tabela a seguir, estão listados todos os operadores C em ordem decrescente de prioridade, com a sua associatividade.

Operadores	Associatividade
() []	esquerda para a direita
! - ++ --	direita para a esquerda
* / %	esquerda para a direita
+ -	esquerda para a direita
< <= > >=	esquerda para a direita

Operadores	Associatividade
== !=	esquerda para a direita
&&	esquerda para a direita
||	esquerda para a direita
= += -= *= /= %=	direita para a esquerda
,	esquerda para a direita

Tabela: Representação da regra de precedência entre operadores.
Sérgio dos Santos Cardoso Silva.

Assim, a expressão a<10 && 5∗b<c é avaliada como (a<10) && ((5∗b)<c).

Confira um exemplo para entender melhor.

Conteúdo interativo

esse a versão digital para executar o código.

Aqui, a condição a>0 && b<0 ‖ c==0 é avaliada em vários momentos devido à precedência dos operadores. Primeiro o a>0 é avaliado, o que será verdadeiro. Em seguida, o b<0 é verificado e também será positivo. Logo, teremos:

```
Positivo && positivo || c == 0
```

A próxima avaliação é o operador &&. Como os dois lados são positivos, o resultado é positivo. Portanto, obtemos:

```
Positivo || c == 0
```

Agora o $c = 0$ é testado e temos como resultado verdadeiro. Com positivo || positivo, o resultado será positivo e será impresso: "A condição é verdadeira".

Primeiro, $agt;0&&blt;0$ é avaliado, e o resultado é então combinado com $c = 00$ usando o operador I|.

Como combinar operadores lógicos com operadores relacionais

Para criar condições mais complexas, você pode combinar operadores lógicos com operadores relacionais. Isso permite que você verifique múltiplas condições ao mesmo tempo. Vamos ver como isso funciona com alguns exemplos.

Verificação de idade e altura

Aqui, estamos combinando operadores relacionais (>=, <=, >) com operadores lógicos (&&) para criar uma condição mais complexa. Veja!

Conteúdo interativo

esse a versão digital para executar o código.

Agora, vamos entender melhor o código anterior.

- A expressão idade >= 18 && idade <= 30 && altura > 1.70 contém três condições:
  - idade >= 18: verifica se a idade é maior ou igual a 18.
  - idade <= 30: verifica se a idade é menor ou igual a 30.
  - altura > 1.70: verifica se a altura é maior que 1.70.

- O operador && é usado para combinar as três condições. A expressão completa será verdadeira apenas se todas as três condições forem verdadeiras.
- Se a expressão for verdadeira, o programa imprime "Você está na faixa etária e tem a altura adequada". Caso contrário, imprime "Você não atende aos critérios".

De acordo com os valores de idade e altura fornecidas, o resultado do programa será “Você está na faixa etária e tem a altura adequada”.

Verificação de temperatura e umidade

No próximo exemplo, verificamos se a temperatura está entre 20 e 30 graus, e se a umidade está acima de 50%.

Conteúdo interativo

esse a versão digital para executar o código.

Agora, vamos compreender melhor o código apresentado.

- A expressão temperatura >= 20.0 && temperatura <= 30.0 && umidade > 50.0 contém três condições:
  - temperatura >= 20.0: verifica se a temperatura é maior ou igual a 20.0 graus.
  - temperatura <= 30.0: verifica se a temperatura é menor ou igual a 30.0 graus.
  - umidade > 50.0: verifica se a umidade é maior que 50.0%.

- O operador && é usado para combinar as três condições. A expressão completa será verdadeira apenas se as três condições forem verdadeiras.
- Se a expressão for verdadeira, o programa imprime "As condições estão favoráveis". Caso contrário, imprime "As condições não estão favoráveis".

Com os valores apresentados, a saída do programa será "As condições estão favoráveis".

Os operadores lógicos são ferramentas essenciais para criar condições complexas em seus programas. Combinando-os com operadores relacionais, é possível verificar múltiplas condições de uma só vez, tornando

o código mais eficiente e dinâmico. Pratique bastante para se familiarizar com esses conceitos e experimente criar suas próprias condições e verificações. Boa codificação!

Estruturas de decisão encadeadas

Aprenda neste vídeo a usar estruturas de decisão encadeadas em C para verificar múltiplas condições sequencialmente. Além disso, entenda a diferença entre estruturas aninhadas e encadeadas.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Diferença entre estruturas aninhadas e encadeadas

Aprenderemos aqui sobre estruturas de decisão encadeadas, um conceito fundamental na programação. É comum confundir estruturas de decisão encadeadas com estruturas de decisão aninhadas, mas elas têm diferenças importantes. Vamos conferir!

Estruturas de decisão aninhadas

Nessa estrutura, uma condição está dentro de outra. Ou seja, temos um if dentro de outro if. Isso é útil para verificar múltiplas condições hierarquicamente. Veja!

```
c

if (condicao1) {
  if (condicao2) {
    // Código a ser executado se condicao1 e condicao2 forem verdadeiras
  }
}
```

Nesse tipo de estrutura, o segundo if só será verificado se o primeiro if for verdadeiro.

Dica

Podem existir diversos níveis de aninhamento, mas é preciso tomar cuidado para que o código não fique confuso.

Estruturas de decisão encadeadas

Nessa estrutura, temos várias condições verificadas sequencialmente, uma após a outra, mas não dentro uma da outra. Usamos else if para criar essas estruturas. Confira!

```
c

if (condicao1) {
  // Código a ser executado se condicao1 for verdadeira
} else if (condicao2) {
  // Código a ser executado se condicao1 for falsa e condicao2 for verdadeira
} else {
  // Código a ser executado se todas as condições anteriores forem falsas
}
```

Cada condição é verificada uma após a outra até que uma seja verdadeira. Quando uma condição é verdadeira, o código correspondente é executado, e as demais condições são ignoradas.

Exemplos práticos de estruturas de decisão encadeadas

Vejamos alguns exemplos para entender melhor como essas estruturas funcionam na prática.

Verificação de idade para classificação etária

Imagine criar um programa que classifique a idade de uma pessoa em categorias: criança, adolescente, adulto ou idoso. Usaremos estruturas encadeadas para isso.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, o programa verifica a idade do usuário e a classifica em diferentes categorias etárias de maneira sequencial, verificando cada condição uma após a outra.

Diante disso, vamos entender melhor o código por meio do seguinte passo a passo:

- A primeira condição if (idade <12) verifica se a idade é menor que 12. Se for verdadeira, o programa imprime "Você é uma criança" e sai da estrutura de decisão.
- Se a primeira condição for falsa, a próxima condição else if (idade >= 12 && idade < 18) é verificada. Se a idade estiver entre 12 e 17, o programa imprime "Você é um adolescente".
- Se a segunda condição também for falsa, a próxima condição else if (idade >= 18 && idade < 60) é verificada. Se a idade estiver entre 18 e 59, o programa imprime "Você é um adulto".
- Se todas as condições anteriores forem falsas, a condição final else é executada, imprimindo "Você é um idoso" para idades de 60 anos ou mais.

Determinação de nota escolar

Aqui, iremos criar um programa que determina a letra da nota de um aluno com base na pontuação numérica. Observe!

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, a nota do aluno é verificada em diferentes intervalos de valores, e o programa imprime a letra correspondente à nota de acordo com a estrutura de decisão encadeada.

Considerando isso, entenda melhor o código por meio do seguinte passo a passo:

- A primeira condição if (nota >= 90) verifica se a nota é maior ou igual a 90. Se for verdadeira, o programa imprime "Sua nota é A" e sai da estrutura de decisão.
- Se a primeira condição for falsa, a próxima condição else if (nota >= 80) é verificada. Se a nota estiver entre 80 e 89, o programa imprime "Sua nota é B".
- Se a segunda condição também for falsa, a próxima condição else if (nota >= 70) é verificada. Se a nota estiver entre 70 e 79, o programa imprime "Sua nota é C".
- Se a terceira condição também for falsa, a próxima condição else if (nota >= 60) é verificada. Se a nota estiver entre 60 e 69, o programa imprime "Sua nota é D".
- Se todas as condições anteriores forem falsas, a condição final else é executada, imprimindo "Sua nota é F" para notas abaixo de 60.

As estruturas de decisão encadeadas são uma maneira eficiente de verificar múltiplas condições de forma sequencial. Elas são usadas quando você tem várias condições que precisam ser verificadas uma após a outra. Ao entender a diferença entre estruturas aninhadas e encadeadas, você poderá escolher a abordagem correta para suas necessidades de programação. Pratique esses conceitos e aplique-os em diferentes cenários para reforçar seu aprendizado.

Estruturas de decisão aninhadas

Aprenda neste vídeo a utilizar estruturas de decisão aninhadas em C para verificar múltiplas condições de forma hierárquica. Além disso, explore a definição e aplicação prática dessas estruturas, acompanhando exemplos claros e detalhados.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Estruturas de decisão aninhadas em C

Agora, vamos explorar as estruturas de decisão aninhadas, um conceito importante para criar programas que respondem de maneira adequada a múltiplas condições hierárquicas.

> As estruturas de decisão aninhadas ocorrem quando uma instrução if está dentro de outra instrução if. Isso é útil quando você precisa verificar uma condição dentro de outra condição.

As estruturas de decisão aninhadas permitem verificar múltiplas condições em diferentes níveis. Se uma condição for verdadeira, outra condição pode ser verificada dentro dela. Confira!

```
c

if (condicao1) {
  if (condicao2) {
    // Código a ser executado se condicao1 e condicao2 forem verdadeiras
  }
}
```

Exemplos de emprego de estruturas de decisão aninhadas

Veja alguns exemplos práticos para entender como funcionam as estruturas de decisão aninhadas.

Verificação de idade e renda

Vamos criar agora um programa que verifica se uma pessoa está qualificada para um desconto especial com base na idade e na renda mensal. A pessoa deve ter mais de 60 anos ou menos de 18 anos e ter uma renda mensal abaixo de 2000.

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

Entenda melhor o passo a passo desse código:

- A primeira condição if (idade > 18 || idade < 60) verifica se a idade do usuário é menor que 18 ou maior que 60. Se for verdadeira, o programa entra na segunda condição.
- A segunda condição if (renda < 2000.0) verifica se a renda do usuário é menor que 2000. Se for verdadeira, o programa imprime "Você está qualificado para o desconto especial".
- Se a segunda condição for falsa, o programa imprime "Você não está qualificado para o desconto devido à renda".
- Se a primeira condição for falsa, o programa imprime "Você não está qualificado para o desconto devido à idade".

Verificação de benefícios sociais

Agora, veremos um exemplo mais complexo, no qual precisamos verificar várias condições para determinar a qualificação para um programa de benefícios sociais. Vamos criar um programa que verifica se uma pessoa é qualificada para benefícios sociais com base em vários critérios: idade, renda e número de dependentes.

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

Entenda melhor o passo a passo desse código:

- A primeira condição if (idade >= 18 && idade <= 65) verifica se a idade do usuário está entre 18 e 65 anos. Se for verdadeira, o programa entra na segunda condição.
- A segunda condição if (renda < 3000.0) verifica se a renda do usuário é menor que 3000. Se for verdadeira, o programa entra na terceira condição.
- A terceira condição if (dependentes > 2) verifica se o número de dependentes é maior que 2. Se for verdadeira, o programa imprime "Você está qualificado para os benefícios sociais".
- Se a terceira condição for falsa, o programa imprime "Você não está qualificado para os benefícios devido ao número de dependentes".
- Se a segunda condição for falsa, o programa imprime "Você não está qualificado para os benefícios devido à renda".
- Se a primeira condição for falsa, o programa imprime "Você não está qualificado para os benefícios devido à idade".

Nesse exemplo, utilizamos estruturas aninhadas para verificar três condições dependentes. Veja!

- Idade

- Renda
- Número de dependentes

As estruturas de decisão aninhadas são úteis quando você precisa verificar múltiplas condições em diferentes níveis. Elas permitem que seu programa tome decisões complexas com base em várias condições dependentes. Pratique esses conceitos e aplique-os em diferentes cenários para reforçar seu aprendizado.

Uso do switch

Aprenda neste vídeo a usar a estrutura switch em C para simplificar a lógica de decisão nos seus programas. Veja a aplicação prática do switch e sua comparação com a estrutura if-else.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução ao switch

A estrutura switch em C é usada para simplificar o processo de seleção de uma entre várias opções possíveis. O switch é especialmente útil quando você tem muitas condições para verificar e deseja evitar uma longa sequência de instruções if-else-if.

> A estrutura switch funciona como um menu. Você fornece uma variável de controle e, com base no valor dessa variável, o programa executa o bloco de código correspondente.

Conheça a sintaxe básica do switch.

Conteúdo interativo

esse a versão digital para executar o código.

Vamos explorar um exemplo prático para entender melhor como usar o switch. Criaremos um programa que apresenta um menu com três opções para o usuário e executa uma ação diferente com base na escolha, veja!

Conteúdo interativo

esse a versão digital para executar o código.

Compreenda melhor esse código com o passo a passo a seguir.

Uso da variável opcao

É utilizada como controle para o switch.

Funcionamento dos cases

Cada case verifica um valor específico da variável opcao. Se o valor da variável corresponder a um dos casos, o código dentro desse case é executado.

Importância do break

É utilizado para sair do switch após a execução do código do case correspondente. Sem o break, o programa continuaria executando os casos seguintes.

Nesse exemplo, se o break fosse removido de todos os cases e o usuário escolhesse a opção 1, o programa executaria todos os cases subsequentes, imprimindo todas as mensagens, pois não há break para interromper a execução. Portanto, adicionar break ao final de cada case é importante para garantir que apenas o código do case correspondente seja executado.

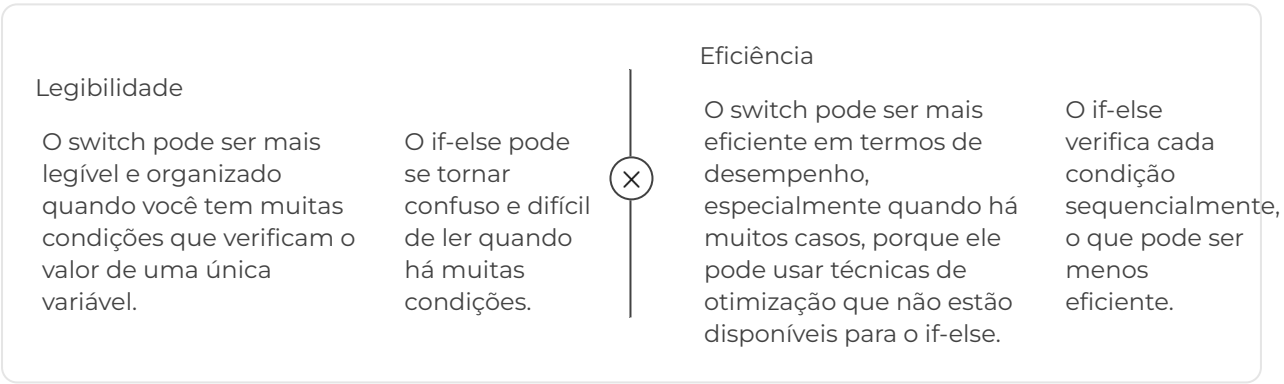
Utilização do default

Embora seja opcional, é uma boa prática incluir o default. Ele fornece um caminho alternativo caso nenhum dos valores especificados nos cases corresponda ao valor da variável de controle. É como um else em uma estrutura if-else, garantindo que sempre haja um bloco de código a ser executado, mesmo que todas as condições falhem.

No exemplo do menu, se o usuário digitar uma opção que não é 1, 2 ou 3, o default será executado, imprimindo "Opção inválida".

Comparação de switch com if-else

Embora o switch e a estrutura if-else possam ser usados para tomar decisões baseadas em condições, existem diferenças importantes. Confira!



A partir disso, veja um exemplo em que um switch pode substituir uma série de if-else criando um programa que imprime o nome do dia da semana com base em um número de 1 a 7.

Assim, observe como o código ficaria empregando o if-else.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, você começa declarando a variável dia do tipo int para armazenar o número do dia da semana. Depois, você decide que hoje é terça-feira e atribui o valor 3 para a variável dia.

O programa então verifica a variável dia usando uma série de instruções if-else-if. Primeiramente, ele verifica se dia é igual a 1. Se for, imprime "Domingo". Como dia é 3, ele continua verificando cada condição else if. Quando ele chega à condição else if (dia == 3), esta é verdadeira, então o programa imprime "Terça-feira". Se dia não for igual a nenhum dos valores de 1 a 7, o else final imprimirá "Dia inválido".

Embora esse método funcione, ele pode se tornar difícil de ler e manter se houver muitas condições. Vamos simplificar isso usando a estrutura switch.

Conteúdo interativo

esse a versão digital para executar o código.

Assim como antes, você declara a variável dia do tipo int para armazenar o número do dia da semana e atribui o valor 3 para indicar que hoje é terça-feira.

A variável dia é usada como controle para o switch. Cada case verifica um valor específico da variável dia. Quando dia é 1, o programa imprime "Domingo". Quando dia é 2, imprime "Segunda-feira". Quando dia é 3, imprime "Terça-feira", e assim por diante.

Atenção

O break em cada case evita que o programa continue a executar os cases seguintes. Isso é importante para garantir que apenas o bloco de código correspondente ao valor de dia seja executado.

Se dia não corresponder a nenhum dos valores nos cases, o default é executado, imprimindo "Dia inválido". Isso funciona como um "else" em uma estrutura if-else, garantindo que sempre haverá uma resposta, mesmo que a entrada do usuário não corresponda a nenhuma das opções.

> O switch torna o código mais organizado e fácil de entender. Cada valor da variável dia é tratado em um case separado, deixando claro o que acontece para cada valor possível.

A estrutura switch é uma ferramenta valiosa para simplificar a lógica de decisão em seus programas. Ela é especialmente útil quando você precisa selecionar uma entre muitas opções com base no valor de uma variável. Comparado com a estrutura if-else, o switch pode tornar seu código mais legível e eficiente. Pratique esses conceitos e experimente usá-los em diferentes cenários para reforçar seu aprendizado.

Menus interativos

Descubra neste vídeo como criar menus interativos em C utilizando toda a estrutura switch. Além disso, veja como implementar opções de escolha em um jogo, gerenciando as ações do usuário de maneira eficiente e clara. Entenda cada passo com exemplos práticos e detalhados.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Criação de menus utilizando switch

Vamos aprender a criar menus interativos em C utilizando a estrutura switch. Os menus interativos são essenciais em muitos programas, pois permitem ao usuário escolher entre várias opções de maneira fácil e organizada. Construiremos um menu simples e aprenderemos como implementar diferentes opções de escolha. Vamos lá!

Implementação de opções de escolha no jogo

Imagine o desenvolvimento de um pequeno jogo em C. Nesse jogo, o usuário terá um menu inicial com opções para iniciar o jogo, ver as regras ou sair. Usaremos a estrutura switch para gerenciar essas opções de maneira clara e eficiente.

Primeiramente, vamos criar um menu que apresenta ao usuário as seguintes opções:

1. Iniciar jogo
2. Ver regras
3. Sair

Considerando essas opções, confira o código básico para esse menu.

Conteúdo interativo

esse a versão digital para executar o código.

Agora, vamos entender melhor o passo a passo desse código.

Declaração da variável

Declaramos a variável opcao do tipo int para armazenar a escolha do usuário.

Exibição do menu

- Usamos printf para exibir o menu principal com três opções.
- Usamos scanf para ler a escolha do usuário e armazená-la na variável opcao.

Estrutura switch

- A variável opcao é usada como controle para o switch.
- Cada case verifica um valor específico da variável opcao. Se o valor da variável corresponder a um dos casos, o código dentro desse case será executado.
- O break é usado para sair do switch após a execução do código do case correspondente. Sem o break, o programa continuaria executando os casos seguintes.
- O default é executado se nenhum dos casos corresponder ao valor da variável. É como um else em uma estrutura if-else. Isso garante que sempre haverá uma resposta, mesmo que a entrada do usuário não corresponda a nenhuma das opções.

Implementação de opções de escolha no jogo

Após entendermos como criar um menu básico, vamos detalhar as ações que cada opção pode executar.

Iniciar jogo

Quando o usuário escolhe a opção 1, o jogo é iniciado. Nesse exemplo, apenas imprimimos a mensagem 'Iniciando o jogo...'. No entanto, você pode adicionar o código necessário para iniciar o jogo aqui.

Ver regras

Quando o usuário escolhe a opção 2, as regras do jogo são exibidas. Usamos **printf** para mostrar uma lista de regras simples. Você pode expandir essa seção para incluir todas as regras detalhadas do seu jogo.

Sair

Quando o usuário escolhe a opção 3, o programa imprime 'Saindo...' e termina a execução.

Opção inválida

Quando o usuário digita um valor que não seja 1, 2 ou 3, o default será executado, apresentando a seguinte mensagem: 'Opção inválida. Tente novamente.' Isso orienta o usuário a fazer uma escolha válida.

Exemplo prático: jogo de adivinhação

Para tornar nosso menu mais interessante, vamos adicionar uma funcionalidade de jogo simples: crie um jogo de adivinhação em que o usuário precisa acertar um número entre 1 e 10.

Conteúdo interativo

esse a versão digital para executar o código.

Aqui estão os detalhes de funcionamento desse código. Acompanhe!

Declaração das variáveis

- Declaramos duas variáveis opcao e palpite do tipo int para armazenar a escolha do usuário e o palpite do número, respectivamente.
- Declaramos a variável numeroSecreto do tipo int para armazenar o número aleatório que o usuário deve adivinhar.

Exibição do menu

- Usamos printf para exibir o menu principal com três opções: Iniciar jogo, Ver regras e Sair.
- Usamos scanf para ler a escolha do usuário e armazená-la na variável opcao.

Estrutura switch

- A variável opcao é usada como controle para o switch.
- Cada case verifica um valor específico da variável opcao. Se o valor da variável corresponder a um dos casos, o código dentro desse case é executado.

Case 1: iniciar jogo

- Quando o usuário escolhe a opção 1, a função srand(time(0)) é chamada para inicializar o gerador de números aleatórios com base no tempo atual. Isso garante que o gerador de números aleatórios produza sequências diferentes cada vez que o programa for executado.
- A variável numeroSecreto é atribuída a um número aleatório entre 1 e 10, gerado pela expressão rand() % 10 + 1. A função rand() gera um número aleatório inteiro, e o operador % 10 limita esse número ao intervalo de 0 a 9. Adicionando 1, obtemos um número entre 1 e 10.
- O programa então solicita ao usuário que adivinhe o número secreto com a mensagem "Adivinhe o número (entre 1 e 10): ".
- Usamos scanf para ler o palpite do usuário e armazená-lo na variável palpite.
- Uma estrutura if-else é usada para comparar o palpite do usuário com o número secreto. Se o palpite for igual ao número secreto, o programa imprime "Parabéns! Você acertou!". Caso contrário, imprime "Você errou. O número era X.", sendo X o número secreto.

Case 2: ver regras

Quando o usuário escolhe a opção 2, o programa imprime as regras do jogo usando printf. As regras incluem informações básicas sobre como jogar.

Case 3: sair

Quando o usuário escolhe a opção 3, o programa imprime "Saindo..." e termina a execução do programa.

Default

Se o usuário digitar um valor que não seja 1, 2 ou 3, o default será executado, imprimindo "Opção inválida. Tente novamente." Isso orienta o usuário a fazer uma escolha válida.

Os menus interativos são uma maneira eficaz de tornar seus programas mais amigáveis e organizados. Utilizando a estrutura switch, você pode gerenciar várias opções de forma clara e eficiente. Experimente criar seus próprios menus interativos e adicione funcionalidades para torná-los mais robustos e interessantes.

Hora de codar

No nível intermediário, você aprendeu sobre operadores lógicos, estruturas de decisão encadeadas e aninhadas, e o uso do switch para criar menus interativos. Agora, vamos aplicar esses conceitos na prática. Desenvolveremos um programa que utiliza menus interativos e lógica complexa para criar um jogo mais dinâmico e funcional. Prepare-se para ver como esses conceitos se combinam para resolver problemas reais de programação.

Confira neste vídeo como aplicar operadores relacionais e lógicos para determinar o vencedor entre o jogador e o computador. Veja esses conceitos na prática!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível aventureiro

Interatividade no Super Trunfo

Neste nível, você dará um grande passo no desenvolvimento do seu Super Trunfo, adicionando interação com o usuário e lógica mais complexa. Continue praticando e se preparando para o desafio final!

O que você vai fazer

Neste desafio, o Super Trunfo fica mais interessante! Você implementará um menu interativo usando switch para que o jogador possa escolher o atributo de comparação entre duas cartas de países. Além disso, você usará estruturas de decisão aninhadas (if-else dentro de if-else) para criar uma lógica de comparação mais complexa, considerando regras específicas para cada atributo. Este desafio é uma continuação do desafio anterior, onde você implementou o cadastro das cartas.

Requisitos funcionais

1. **Menu Interativo:** Criar um menu interativo no terminal usando a estrutura switch que permita ao jogador escolher qual atributo será usado para comparar as cartas. O menu deve ser claro e fácil de usar.

2. **Comparação de Atributos:** Implementar a lógica de comparação entre duas cartas com base no atributo selecionado pelo jogador. Os atributos disponíveis são:
    - Nome do país (string - usado apenas para exibir informações, não para comparação direta)
    - População (int)
    - Área (float)
    - PIB (float)
    - Número de pontos turísticos (int)
    - Densidade demográfica (float - já calculada no desafio anterior).
3. **Regras de Comparação:** A regra geral é: vence a carta com o **maior** valor no atributo escolhido. Porém, para a **Densidade Demográfica**, a regra inverte: vence a carta com o **menor** valor.
4. **Exibição do Resultado:** Mostrar na tela, de forma clara, o resultado da comparação, incluindo:
    - O nome dos dois países.
    - O atributo usado na comparação.
    - Os valores do atributo para cada carta.
    - Qual carta venceu.
    - Em caso de empate, exibir a mensagem "Empate!".

Requisitos não funcionais

- **Usabilidade:** O menu e as mensagens exibidas no terminal devem ser intuitivos e fáceis de entender.
- **Performance:** O sistema deve responder rapidamente às ações do usuário.
- **Manutenibilidade:** Escreva um código limpo, organizado e bem comentado.
- **Segurança:** (Embora menos crítico neste nível, comece a pensar em como seu código poderia lidar com entradas inválidas do usuário, como a escolha de uma opção inexistente no menu. Um default no switch pode ajudar).

Simplificações para o nível intermediário

- Você pode usar as cartas que já foram cadastradas no desafio anterior. Não é necessário implementar o cadastro novamente neste nível.
- Foque na criação do menu com switch e na lógica de comparação com if-else (incluindo comparações aninhadas onde fizer sentido).
- Implemente a comparação para apenas **duas cartas**.

Entregando seu projeto

1. Desenvolva seu projeto no GitHub em um repositório público.

2. Certifique-se de que seu código está bem comentado e que o arquivo README.md do seu repositório contém instruções claras de como compilar e executar seu programa. Inclua exemplos de como usar o menu e quais são os atributos disponíveis para comparação.

3. Envie o link do seu repositório do GitHub no SAVA.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Entendendo o operador ternário em C

Confira neste vídeo como usar o operador ternário em C, uma forma compacta de escrever instruções if-else. Aprenda a utilizar essa ferramenta eficaz para simplificar seu código, com exemplos práticos.

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

Definição e aplicação do operador ternário

O operador ternário é uma forma compacta de escrever uma instrução if-else. Ele é chamado de ternário porque envolve três partes. Vamos conhecê-las!

- Uma condição
- Um valor se a condição for verdadeira
- Um valor se a condição for falsa

A partir disso, vamos conhecer a sintaxe do operador ternário.

```
condicao ? valor_se_verdadeiro : valor_se_falso;
```

O operador ternário é utilizado quando queremos tomar uma decisão rápida baseada em uma única condição. É uma maneira elegante de simplificar o código, especialmente quando a lógica é simples. Veja a seguir algumas situações para entender melhor seu uso.

Exemplos de aplicação do operador ternário

Verificação de idade

Imagine imprimir "Maior de idade" se a idade for 18 ou mais, e "Menor de idade" caso contrário. Podemos fazer isso com o seguinte operador ternário.

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

Nesse exemplo, declaramos uma variável idade com o valor 20. Usamos o operador ternário para verificar se idade é maior ou igual a 18. Se for, resultado recebe a string "Maior de idade"; caso contrário, recebe "Menor de idade". O programa então imprime o resultado que, nesse caso, será “Maior de idade”.

Verificação de temperatura

Aqui observamos se uma temperatura está superior ou inferiro a certo valor. Por exemplo, se a temperatura está superior a 30 graus Celsius.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse caso, declaramos a variável temperatura com o valor 28. Usamos o operador ternário para decidir se a variável estado deve ser "Calor" ou "Frio" com base na temperatura. Como 28 não é maior que 30, estado recebe "Frio". O programa então imprime "Estado: Frio".

Determinação de maior número

Outro exemplo útil é determinar o maior de dois números. Vamos fazer isso utilizando o seguinte operador ternário.

Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, temos duas variáveis, num1 e num2, com os valores 10 e 20, respectivamente. Usamos o operador ternário para verificar qual dos dois números é maior. Se num1 for maior que num2, maior recebe o valor de num1; caso contrário, recebe o valor de num2. O programa imprime 'O maior número é: 20'.

O operador ternário é uma ferramenta útil em C que permite simplificar instruções if-else em uma única linha de código. O operador ternário torna o código mais conciso e legível, especialmente quando a lógica de decisão é simples.

Agora que você compreendeu como utilizar o operador ternário, experimente integrá-lo em seus próprios projetos para tornar seu código mais eficiente e elegante. Lembre-se, a prática é essencial para dominar esse e outros conceitos avançados de programação.

Integração de estruturas de decisão

Confira neste vídeo como integrar estruturas de decisão como if, if-else e switch em um programa C. Aprenda a desenvolver um sistema de gerenciamento de estudantes para calcular médias e determinar o status.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para um bom desenvolvedor é importante saber combinar diferentes estruturas de decisão para criar programas com lógica mais robusta e eficiente. As estruturas **if**, **if-else** e **switch** são fundamentais para tomar decisões baseadas em condições variadas.

Na sequência, vamos explorar como integrar essas estruturas em um único programa para resolver problemas complexos. Desenvolveremos um programa que gerencia as notas de estudantes e determina seu status (aprovado, em recuperação ou reprovado) com base nas notas. O programa também terá um menu para o usuário escolher entre diferentes operações.

Conteúdo interativo

esse a versão digital para executar o código.

Vamos analisar a lógica desse programa passo a passo.

Declaração das variáveis

- Declaramos a variável opcao do tipo int para armazenar a escolha do usuário.
- Declaramos as variáveis nota1, nota2 e media do tipo float para armazenar as notas dos estudantes e a média calculada.

Exibição do menu

- Usamos printf para exibir o menu de gerenciamento de estudantes.
- Usamos scanf para ler a escolha do usuário e armazená-la na variável opcao.

Estrutura switch

- A variável opcao é usada como controle para o switch.
- Cada case verifica um valor específico da variável opcao. Se o valor da variável corresponder a um dos casos, o código dentro desse case é executado.
- O break é usado para sair do switch após a execução do código do case correspondente. Sem o break, o programa continuaria executando os casos seguintes.
- O default é executado se nenhum dos casos corresponder ao valor da variável. Isso garante que sempre haverá uma resposta, mesmo que a entrada do usuário não corresponda a nenhuma das opções.

Cálculo da média (case 1)

- Quando o usuário escolhe a opção 1, o programa solicita as duas notas do estudante.
- Calculamos a média das notas e armazenamos na variável media.
- Usamos printf para exibir a média calculada.

Determinação do status (case 2)

- Quando o usuário escolhe a opção 2, o programa solicita a média do estudante.
- Usamos a estrutura if-else para determinar o status do estudante com base na média:
  - Se a média for maior ou igual a 7.0, o status é "Aprovado".
  - Se a média for maior ou igual a 5.0, mas menor que 7.0, o status é "Recuperação".
  - Se a média for menor que 5.0, o status é "Reprovado".
- Usamos printf para exibir o status determinado.

Saída do programa (case 3)

Quando o usuário escolhe a opção 3, o programa imprime "Saindo..." e termina a execução do programa.

Opção inválida (default)

Se o usuário digitar um valor que não seja 1, 2 ou 3, o default será executado, imprimindo "Opção inválida. Tente novamente." Isso orienta o usuário a fazer uma escolha válida.

Integrar estruturas de decisão como if, if-else e switch em um único programa permite criar lógicas mais robustas e eficientes. No exemplo do sistema de gerenciamento de estudantes, vimos como usar essas estruturas para calcular a média das notas e determinar o status do estudante com base nas notas.

Resumindo

A combinação dessas estruturas torna o programa mais dinâmico e interativo, oferecendo uma melhor experiência ao usuário. Experimente aplicar essas técnicas em seus próprios projetos para desenvolver soluções mais sofisticadas e funcionais.

Hora de codar

É hora de aplicar os conceitos avançados estudados para desenvolver um jogo de Maior, Menor ou Igual contra o computador. Utilizando operadores ternários, estruturas de decisão if, if-else e switch, você criará um programa em que o jogador escolhe um número e um tipo de comparação (maior, menor ou igual) para competir contra um número gerado aleatoriamente pelo computador. Esse exercício consolidará seu entendimento de como integrar diferentes estruturas de decisão para criar lógicas mais complexas e dinâmicas.

Confira neste vídeo como desenvolver um jogo de Maior, Menor ou Igual em C. Aprenda a utilizar operadores ternários e estruturas de decisão if, if-else e switch. Além disso, veja como integrar esses conceitos para criar um programa interativo e funcional.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível mestre

Implementando Comparações Avançadas com Atributos Múltiplos em C

Este é o desafio final do Super Trunfo! Coloque em prática tudo o que você aprendeu e mostre suas habilidades de programação!

O que você vai fazer

Prepare-se para o desafio final do Super Trunfo!Agor você integrará tudo o que aprendeu sobre estruturas de decisão em C para criar uma lógica de comparação ainda mais sofisticada. Você permitirá que o jogador escolha **dois atributos** para comparar as cartas de países, implementará menus dinâmicos com switch e usará o operador ternário para tornar seu código mais elegante. Este desafio é a culminação dos desafios anteriores, onde você cadastrou as cartas e implementou a comparação com um único atributo.

Requisitos funcionais

1. **Escolha de Dois Atributos:** O sistema deve permitir que o jogador escolha **dois atributos numéricos diferentes** para a comparação das cartas, através de menus interativos. Implemente a lógica necessária para garantir que o jogador não possa selecionar o mesmo atributo duas vezes.

2. **Comparação com Múltiplos Atributos:** Implemente a lógica para comparar as duas cartas com base nos dois atributos escolhidos. Para cada atributo, a regra geral é: vence a carta com o **maior** valor. A exceção continua sendo a **Densidade Demográfica**, onde vence a carta com o **menor** valor.

3. **Soma dos Atributos:** Após comparar os dois atributos individualmente, o sistema deve **somar os valores** dos atributos para cada carta. A carta com a **maior soma** vence a rodada.

4. **Tratamento de Empates:** Implemente a lógica para tratar empates. Se a soma dos atributos das duas cartas for igual, exiba a mensagem "Empate!".

5. **Menus Dinâmicos:** Crie menus interativos e dinâmicos usando switch. "Dinâmicos" significa que, por exemplo, após o jogador escolher o primeiro atributo, esse atributo não deve mais aparecer como opção para a escolha do segundo atributo.

6. **Exibição Clara do Resultado:** Mostre o resultado da comparação de forma clara e organizada, incluindo:
    - O nome dos dois países.
    - Os dois atributos usados na comparação.
    - Os valores de cada atributo para cada carta.
    - A soma dos atributos para cada carta.
    - Qual carta venceu (ou se houve empate).

Requisitos não funcionais

- **Usabilidade:** A interface do usuário deve ser intuitiva e fácil de navegar, com mensagens claras e instruções para cada etapa.

- **Performance:** O sistema deve executar as comparações e exibir os resultados rapidamente.
- **Manutenibilidade:** Escreva um código bem estruturado, organizado, com indentação consistente e comentários explicativos. Use nomes de variáveis descritivos.
- **Confiabilidade:** O sistema deve ser robusto e lidar com entradas inválidas do usuário de forma adequada, exibindo mensagens de erro apropriadas e evitando travamentos. Utilize o default no switch para tratar opções inválidas.

Simplificações para o nível avançado

- Você pode continuar usando as cartas pré-cadastradas dos desafios anteriores. Não precisa reimplementar o cadastro.
- Foque na integração de todos os elementos: menus dinâmicos com switch, comparações com if-else (e aninhamentos), operador ternário, cálculo da soma dos atributos e tratamento de empates.
- A comparação é feita entre apenas **duas cartas**.

Entregando seu projeto

1. Desenvolva seu projeto no GitHub, em um repositório público.
2. Assegure-se de que seu código esteja bem comentado e que o arquivo README.md no seu repositório contenha instruções claras de como compilar e executar o programa. Inclua exemplos de uso dos menus e os atributos disponíveis para comparação.
3. Submeta o link do repositório do seu projeto no GitHub via SAVA.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Considerações finais

Você adquiriu uma compreensão profunda de como tomar decisões em seus programas utilizando diferentes estruturas de decisão. Além disso, aplicou esses conceitos na prática, desenvolvendo jogos e outras soluções interativas.

Vamos revisar os conteúdos mais relevantes que você explorou ao longo deste conteúdo e refletir sobre o que foi aprendido.

1. **Operadores relacionais e lógicos:** aprendemos a usar operadores relacionais para comparar valores e operadores lógicos para combinar condições. Esses operadores são fundamentais para tomar decisões baseadas em múltiplas condições.

2. **Estruturas de decisão simples e compostas:** utilizamos if e if-else para criar lógicas de decisão simples e compostas, permitindo ao programa responder de maneiras diferentes com base nas condições verificadas.

3. **Estruturas de decisão aninhadas e encadeadas:** exploramos como combinar múltiplas condições de forma aninhada e encadeada, aumentando a complexidade e a precisão das decisões tomadas pelo programa.

4. **Operador ternário:** essa ferramenta oferece uma maneira mais concisa de escrever instruções if-else, simplificando o código para decisões rápidas.

5. **Uso do switch:** aprendemos a usar a estrutura switch para simplificar a seleção entre múltiplas opções, tornando o código mais claro e fácil de manter.

6. **Integração de estruturas de decisão:** no nível avançado, combinamos diferentes estruturas de decisão (if, if-else, switch) para criar lógicas complexas e eficientes, aplicando esses conceitos em um jogo interativo.

Referências

DAMAS, L. **Linguagem C.** 10. ed. Rio de Janeiro: LTC, 2024.

SCHILDT, H. **C completo e total.** 3. ed. São Paulo: Makron Books, 1996.

KERNIGHAN, B. W.; RITCHIE, D. M. **The C Programming Language.** 2. ed. Englewood Cliffs: Prentice Hall, 1988.

DEITEL, P.; DEITEL, H. **C: How to Program.** 8. ed. Boston: Pearson, 2015.