



Super Trunfo em C: fundamentos e técnicas avançadas

Você irá desenvolver um programa em C que crie e gerencie dados das cartas para o jogo de Super Trunfo, com o propósito de aprimorar habilidades práticas na manipulação de variáveis, operadores e funções de entrada e saída.

Curadoria de TI

Objetivos

- Implementar um programa em C para cadastrar cartas do jogo Super Trunfo de Países, permitindo a entrada e exibição de informações básicas das cidades.
- Aplicar operadores matemáticos em C para aprimorar as funcionalidades do jogo Super Trunfo de Países.
- Aplicar operadores relacionais e modificadores de tipo em C para refinar o jogo Super Trunfo de Países.

Introdução

Prepare-se para mergulhar no mundo do desenvolvimento de jogos! Neste desafio, você assumirá o papel de programador(a) na TechNova, uma empresa líder em inovação tecnológica. A TechNova está dando os primeiros passos na criação de um jogo de Super Trunfo com o tema "Países" e contratou você para desenvolver a parte inicial do projeto: o sistema de cadastro de cartas. Ao longo de três módulos, você transformará requisitos em código funcional, aprendendo na prática e construindo uma base sólida em programação C.

Descubra como a teoria se funde à prática enquanto avançamos juntos em cada fase. Desvendaremos a proposta, exploraremos a metodologia e celebraremos o incrível resultado que você alcançará ao final de cada desafio!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Cenário

A TechNova, gigante da inovação tecnológica, está construindo um jogo de Super Trunfo com o tema "Países" e você foi o(a) escolhido(a) para desenvolver o sistema inicial de cadastro de cartas.

No Super Trunfo, jogadores competem usando cartas com diferentes atributos. Um jogador escolhe um atributo de sua carta e o outro jogador compara o mesmo atributo de sua própria carta. Quem tiver o valor mais alto para o atributo escolhido vence a rodada e leva as cartas do adversário

Sua missão?

Criar um programa em C que permita registrar cada carta com informações detalhadas sobre as cidades, utilizando variáveis, operadores e funções de entrada e saída. Cada carta conterá dados essenciais como: estado, código, nome da cidade, população, PIB, área e número de pontos turísticos. Além disso, você também calculará e adicionará propriedades derivadas, como densidade populacional e PIB per capita, enriquecendo as informações estratégicas de cada carta.

Conceitos de solução estruturada

Assista ao vídeo para aprender a criar programas bem estruturados e organizados em C. Confira os conceitos fundamentais de programação estruturada, a importância da organização e clareza no código, e como o raciocínio lógico e o pensamento computacional ajudam a resolver problemas de forma eficiente.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução ao conceito de solução estruturada

O computador não tem discernimento para julgar se algo é certo ou errado. No entanto, ele pode ser instruído a tomar decisões lógicas, baseadas em critérios objetivos, e seguir instruções ordenadas. Isso é o que chamamos de **solução estruturada**.

Assim, ao começarmos a aprender a programar, uma das primeiras coisas que precisamos entender é o conceito de solução estruturada. Mas o que isso significa?



Reflexão

Imagine que você tenha uma receita para fazer um bolo. Essa receita é dividida em passos simples e claros: misture os ingredientes, asse a massa, e assim por diante.

Programar é parecido com seguir uma receita. Dividimos um problema grande em partes menores e mais gerenciáveis, o que torna a resolução mais fácil e organizada. Voltando ao exemplo do bolo, na cozinha, isso é o que chamamos de *mise en place*. Se os ingredientes e as etapas não forem organizados, o resultado pode ser um bolo malfeito.



Bolo malfeito.

Importância da organização e clareza no código

Imagine que você tenha encontrado a receita de um bolo, mas ela esteja toda desorganizada. Um ingrediente está listado no meio das instruções, e a temperatura do forno está no final do texto. Difícil de seguir, não é? O mesmo acontece com o código de um programa. Um código bem organizado facilita a leitura e a manutenção não só para você, mas também para outras pessoas que possam trabalhar no seu projeto.

Para manter o código organizado e claro, podemos usar algumas práticas!

Indentação

Espaço extra adicionado no início de uma linha de código para indicar que ela está dentro de um bloco de código. Isso ajuda a visualizar blocos de código e entender a hierarquia das instruções. Veja como fica o código sem indentação. Não se preocupe em entender a finalidade dele, é apenas para você entender o conceito.

```
c

#include
int main() {int num1, num2, soma;printf("Digite o primeiro número: ");scanf("%d",
&num1);printf("Digite o segundo número: ");scanf("%d", &num2);soma = num1 +
num2;printf("A soma de %d e %d é: %d\n", num1, num2, soma);return 0;}
```

Agora, veja o mesmo código com indentação!

```
c

#include

int main() {
    int num1, num2, soma;

    // Lendo dois números do usuário
    printf("Digite o primeiro número: ");
    scanf("%d", &num1);

    printf("Digite o segundo número: ");
    scanf("%d", &num2);

    // Calculando a soma dos dois números
    soma = num1 + num2;

    // Exibindo o resultado
    printf("A soma de %d e %d é: %d\n", num1, num2, soma);

    return 0;
}
```

O segundo código não é mais fácil de visualizar?

A indentação facilita a leitura e a compreensão do código, pois é possível perceber visualmente quais linhas pertencem a que bloco de código.

Comentários

São uma ferramenta essencial na programação, especialmente na linguagem C, pois ajudam a documentar o código, tornando-o mais legível e compreensível para outros programadores e para o próprio autor do código no futuro. Eles são ignorados pelo compilador, o que significa que não afetam a execução do programa, mas fornecem informações valiosas sobre a lógica e a estrutura do código.

Existem dois tipos principais de comentários na linguagem C. Veja!

Linguagem única

Começam com `//` e se estendem até o final da linha. Eles são frequentemente usados para breves anotações ou para explicar uma linha específica de código.

```
int x = 10; // Declara uma variável inteira e a inicializa com o valor 10.
```

Múltiplas linhas

Começam com `/*` e terminam com `*/`. Eles são usados para descrever seções maiores de código ou fornecer explicações detalhadas.

```
/*
```

Essa função calcula a soma de dois números inteiros.

Parâmetros:

- int a: o primeiro número

- int b: o segundo número

Retorno:

- int: a soma de a e b

```
*/
```

```
int soma(int a, int b) {
```

```
    return a + b;
```

```
}
```

Nomes significativos

Use nomes de variáveis e funções que façam sentido. Por exemplo, em vez de `x` e `y`, use `soma` ou `resultado`. Você irá compreender melhor mais à frente!

Raciocínio lógico e pensamento computacional

O raciocínio lógico é a capacidade de pensar de maneira clara e ordenada. Na programação, usamos o raciocínio lógico para resolver problemas de forma sistemática. Já o pensamento computacional é um conjunto de habilidades que nos permite abordar problemas de maneira que um computador possa entender.

Para desenvolver essas habilidades, podemos seguir alguns passos. Acompanhe!



Programador focado em sua atividade profissional.

1

Decomposição

Consiste em dividir o problema inicial em partes menores, permitindo que cada parte menor seja mais facilmente resolvida. Geralmente, problemas que não estão decompostos têm sua resolução mais difícil de enxergar e desenvolver. Por exemplo, se você quer calcular a média de notas, pode dividir em: ler as notas, somar as notas, dividir pelo número de notas.

Reconhecimento de padrões

Ao decompor um problema em partes menores, fica mais fácil focar os detalhes e perceber que algumas dessas partes já são conhecidas ou têm soluções conhecidas. O reconhecimento de padrões envolve identificar repetições ou regras, o que pode ajudar a resolver problemas similares. Por exemplo, se você sabe calcular a média de três números, pode usar a mesma lógica para calcular a média de cinco números ou a média das temperaturas de uma semana.

3

Abstração

Consiste na filtragem e classificação dos dados, concentrando a atenção no que realmente é importante. Ela pode ser vista como o principal dos pilares, pois será utilizada em diversos momentos. Ao concentrar-se nos dados realmente importantes, a abstração permite que decisões sejam tomadas com maior qualidade. Por exemplo, ao calcular a média, não importa o nome do aluno, apenas as notas.

4

Algoritmos

São procedimentos para resolver um problema, detalhando as ações a serem executadas e a ordem em que devem ocorrer. Um algoritmo é como uma receita de bolo: uma lista de instruções claras e precisas. Por exemplo, uma lista resumida de passos para fazer um bolo seria:

- Reúna os ingredientes
- Misture os ingredientes
- Prepare a forma
- Asse o bolo
- Deixe esfriar e sirva.

Compreender e aplicar o conceito de solução estruturada é muito importante para se tornar um bom programador. Manter o código organizado e claro não só facilita sua vida, mas também ajuda outras pessoas que possam trabalhar no seu projeto. Desenvolver raciocínio lógico e pensamento computacional é um processo contínuo, e a prática constante é a chave para a melhoria.

Processo de programação e linguagem C

Confira no vídeo o processo de programação e os fundamentos da linguagem C. Entenda como a programação funciona, desde a escrita do código-fonte até a execução do programa, e familiarize-se com os elementos básicos que compõem a linguagem C.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Processo de programação

Vamos falar sobre o processo de programação e entender melhor a linguagem C. Programar é como dar instruções para um computador fazer o que você quer, como se estivesse ensinando alguém a seguir uma receita. Programar envolve várias etapas. Vejamos cada uma delas!

Entendimento do problema

Primeiramente, você precisa entender o que deve ser resolvido. Para fazer um bolo, por exemplo, você precisa saber quais ingredientes usar, como misturar os ingredientes, como assar e o que o bolo deve ter no final.

Planejamento

Aqui, você decide como resolver o problema. Para o bolo, você decide a ordem dos passos: misturar os ingredientes, assar e decorar. Em nosso caso, vamos planejar o código usando ferramentas como pseudocódigo e fluxogramas.

Codificação

Agora, escrevemos o código. É como seguir a receita escrita, mas usando uma linguagem que permita ser traduzida para códigos que o computador consiga compreender e executar todos os passos. Essa linguagem é a chamada linguagem de programação.

Compilação

Essa é a etapa que traduz o código que você escreveu na linguagem C para uma linguagem que o computador entende. É como se fosse o processo de pegar todos os ingredientes, misturar de acordo com a receita e ter a massa que será levada ao forno para assar.

Teste

Esse é o momento em que verificamos se tudo funciona. No caso do bolo, seria provar para ver se está gostoso. Para o código, rodamos para ver se ele faz o que deveria fazer.

Depuração

Se algo deu errado, corrigimos os erros. No código, você faz ajustes até tudo funcionar perfeitamente.

Manutenção

Depois de tudo pronto, pode ser necessário fazer ajustes ou melhorias. É como armazenar a receita para usá-la de novo ou melhorar o próximo bolo.

O que é uma linguagem de programação?

Uma linguagem de programação é um conjunto de instruções que você pode usar para comunicar-se com um computador. Pense em uma linguagem de programação como um idioma. Assim como utilizamos o português para nos comunicar com outras pessoas, usamos linguagens de programação para comunicar nossos desejos aos computadores. Algumas linguagens são melhores para certas tarefas. C é uma dessas linguagens, e é ótima para aprender os fundamentos de programação.



Programador escrevendo instruções em linguagem de programação.

As linguagens de programação foram desenvolvidas para facilitar a escrita de programas de computador, fornecendo uma maneira padronizada para que os programadores escrevam instruções que podem ser traduzidas em ações que um computador pode executar. Existem diferentes tipos de linguagens de programação, cada uma com seu propósito específico e nível de complexidade. Algumas são projetadas para tarefas específicas, como manipulação de dados científicos, enquanto outras são mais gerais e podem ser usadas para uma gama de aplicações.

Conheça agora as características das linguagens de programação.

Sintaxe e semântica

Cada linguagem tem sua própria sintaxe (regras sobre como escrever o código) e semântica (significado das instruções).

Nível de abstração

As linguagens podem ser de alto nível (mais próximas da linguagem humana) ou de baixo nível (mais próximas da linguagem de máquina).

Paradigmas de programação

Diferentes linguagens suportam diferentes estilos de programação, como programação orientada a objetos, programação funcional, programação procedural etc.

Por que usar C?

Empregar a linguagem C nos traz as seguintes vantagens. vejamos!

Eficiência

C é muito rápido e eficiente, o que é importante para programas que precisam executar rapidamente.

Versatilidade

Você pode usar C para quase qualquer tipo de software, desde sistemas operacionais até jogos.

Fundamento sólido

Aprender C ajuda você a entender como os computadores funcionam de forma mais profunda, já que muitas outras linguagens foram baseadas em C.

Processo de tradução

Quando escrevemos código em C, estamos usando uma linguagem que é mais fácil para nós, humanos, entendermos. Mas o computador não fala C diretamente. Precisamos traduzir esse código para que o computador possa executar. Aqui estão, de forma resumida, os passos desse processo. Confira!

Escrita do código-fonte

Escrevemos nosso programa em um arquivo de texto com uma linguagem de programação, como C. Esse arquivo é chamado de código-fonte e, para a linguagem C, tem a extensão ".c". Por exemplo, "meu_programa.c".

Compilação

Usamos um compilador, um programa especial que converte o código-fonte em linguagem de máquina que o computador pode entender, criando um arquivo binário executável. Por exemplo, "meu_programa.exe" no Windows ou "meu_programa" no Linux.

Ligação (Linking)

Muitas vezes, nosso código precisa de outras bibliotecas ou recursos. O linker combina nosso código compilado com essas bibliotecas para criar o arquivo executável final.

Execução

Executamos o arquivo binário no computador, e ele fará o que programamos.

Características da linguagem C

C é uma linguagem poderosa e flexível, com algumas características importantes. Vamos conhecê-las!

1

Sintaxe simples

A estrutura da linguagem C é simples e direta, o que facilita a leitura e escrita do código.

2

Portabilidade

Programas escritos em C podem ser executados em diferentes tipos de computadores com pouca ou nenhuma modificação.

3 Controle baixo nível

C permite acesso direto à memória e manipulação de hardware, algo que não é possível em algumas linguagens.

4

Bibliotecas ricas

C possui uma vasta coleção de bibliotecas que fornecem funções prontas para uma variedade de tarefas.

Exemplos de um código escrito em C

O código a seguir solicita ao usuário que ele insira dois números, realize a soma dos dois números e apresente o resultado na tela. Você ainda deve achar tudo muito confuso, mas não se preocupe. Ao longo deste conteúdo você vai compreender o que faz cada uma dessas instruções. O objetivo agora é apenas você matar sua curiosidade de ver um código-fonte.

```
c
#include

int main() {
    int num1, num2, soma;

    // Lendo dois números do usuário
    printf("Digite o primeiro número: ");
    scanf("%d", &num1);

    printf("Digite o segundo número: ");
    scanf("%d", &num2);

    // Calculando a soma dos dois números
    soma = num1 + num2;

    // Exibindo o resultado
    printf("A soma de %d e %d é: %d\n", num1, num2, soma);

    return 0;
}
```

Meu primeiro programa em C

Neste vídeo, você aprenderá a criar seu primeiro programa em C. Descubra como configurar o ambiente de desenvolvimento, compilar e executar um simples "Olá, Mundo!" na tela. Veja como configurar uma IDE e compilar seu código no Windows e no Linux, preparando-o para escrever e executar seus próprios programas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Criação de um programa simples

Agora que você já conhece os conceitos fundamentais de programação, vamos colocar a mão na massa e criar seu primeiro programa em C.

Vamos criar um programa simples que exibe a mensagem "Olá, Mundo!" na tela. Esse é o programa tradicional de introdução em muitas linguagens de programação.

Escrevendo o código

Aqui está um exemplo de um programa básico em C que está inserido em um compilador on-line. Para ver seu primeiro programa em execução, clique em Executar e veja o resultado!



Conteúdo interativo

esse a versão digital para executar o código.

Esse código é bem simples. Ele tem apenas três partes principais:

- **Inclusão da biblioteca** : `#include` inclui a biblioteca padrão de entrada e saída, necessária para usar a função `printf`.
- **Função principal** : `int main()` é onde o programa começa a execução. Toda aplicação em C começa pela função `main`.
- **Saída de dados** : A linha `printf("Olá, Mundo!\n");` imprime a mensagem "Olá, Mundo!" na tela. O `\n` adiciona uma nova linha ao final da mensagem.
- **Fim do programa** : `return 0;` indica que o programa terminou com sucesso.

Compilação e execução do programa

Para que o computador entenda e execute o código, precisamos compilá-lo. Você precisará de uma IDE (ambiente de desenvolvimento integrado) e um compilador instalados no seu computador. Algumas opções populares incluem Code::Blocks, Dev-C++, e Visual Studio Code com extensões adequadas.



Dica

Uma opção mais simples é criar um arquivo texto, usando um editor de texto simples, instalar o compilador e realizar o processo de compilação e execução em um terminal.

Você também pode usar o GitHub Codespaces, que fornece um ambiente de desenvolvimento on-line.

Veja a seguir os passos básicos para configuração do seu ambiente de trabalho para algumas plataformas.

Windows

Aqui estão os passos gerais para instalação no Windows. Confira!

Instalação

Certifique-se de que você tem um compilador como MinGW instalado. Você pode baixar o MinGW em mingw-w64.org.

Instalação de uma IDE

- **Code::Blocks:** baixe e instale o Code::Blocks com o compilador MinGW incluído.
- **Dev-C++:** Outra opção é o Dev-C++.
- **Visual Studio Code:** baixe e instale o Visual Studio Code, e adicione a extensão C/C++ da Microsoft. Você também precisará do compilador MinGW.

Escrevendo o código

Digite o código acima na IDE escolhida e salve o arquivo com a extensão .c, por exemplo, meu_programa.c. Ou digite o código acima em um editor de texto e salve o arquivo com a extensão .c, por exemplo, meu_programa.c.

Compilação

Dependendo do ambiente que você configurou, use um dos seguintes passos:

- **Code::Blocks:** No Code::Blocks, clique em "Build and Run" para compilar e executar o programa.
- **Dev-C++:** No Dev-C++, clique em "Compile & Run".

Linux

Aqui estão os passos gerais para instalação no Linux.

GNU Compiler Collection é um conjunto de compiladores de linguagens de programação. Siga os passos para utilizá-lo:

Instalação

A maioria das distribuições Linux já vem com o GNU Compiler Collection (gcc) instalado. Se não estiver, você pode instalar usando o gerenciador de pacotes da sua distribuição. Por exemplo, no Ubuntu, você pode usar:

```
sudo apt-get install gcc
```

Escrevendo o código

Digite o código acima em um editor de texto e salve o arquivo com a extensão .c, por exemplo, meu_programa.c.

Compilação

Abra o terminal e navegue até o diretório onde seu arquivo está salvo. Use o gcc para compilar o código:

```
gcc -o meu_programa meu_programa.c
```

Isso cria um arquivo executável chamado meu_programa.

Execução

Execute o programa digitando:

```
./meu_programa
```

GitHub Codespaces

A **GitHub Codespaces** é uma plataforma de desenvolvimento em nuvem. Siga os passos para utilizá-la:

Acessando Codespaces

Vá até o repositório no GitHub e clique em "Code" > "Open with Codespaces". Isso abrirá um ambiente de desenvolvimento on-line.

Escrevendo o código

No Codespaces, crie um arquivo com a extensão .c, por exemplo, meu_programa.c, e digite o código.

Compilação e execução

No terminal integrado do Codespaces, compile e execute o programa usando os comandos gcc e ./meu_programa.

Com essas explicações e exemplos, você deve estar pronto para criar e executar seu próprio programa em C. Pratique escrevendo o código, compilando e executando, e observe como pequenas mudanças no código podem alterar o comportamento do programa.

Variáveis e tipos de dados

Assista ao vídeo para aprender sobre variáveis e tipos de dados. Veja como declarar e inicializar variáveis, entender os tipos de dados primitivos como int, float, double e char, e as regras para criar nomes de variáveis.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vamos falar sobre variáveis e tipos de dados, dois conceitos fundamentais na programação.

O que são variáveis?

Pense nas variáveis como caixas onde você pode armazenar informações. Cada caixa tem um nome, e você pode usar esse nome para colocar algo dentro dela ou pegar o que está lá. Em programação, essas caixas são usadas para guardar valores que podem mudar enquanto o programa está rodando.

Declaração de variáveis

Declarar uma variável é como criar uma caixa e dar um nome a ela. Em C, a declaração de variáveis segue um formato específico, definindo primeiro o tipo de variável e depois o nome, isso é o que chamamos de sintaxe da declaração de variáveis.

```
Tipo_variável nome_variável
```

Vamos ver exemplos de declaração de variáveis:

- `int idade; // Declara uma variável inteira chamada "idade"`
- `float altura; // Declara uma variável de ponto flutuante chamada "altura"`
- `double salario; // Declara uma variável de ponto flutuante de dupla precisão chamada "salario"`
- `char opcao; // Declara uma variável de caractere chamada "opcao"`

Aqui, estamos dizendo ao computador que queremos quatro caixas: uma para armazenar um número inteiro chamado `idade`, uma para armazenar um número com casas decimais chamado `altura`, outra para armazenar um número com mais casas decimais chamado `salario`, e uma última para armazenar um único caractere chamado `opcao`.

Regras para criação de nomes de variáveis em C

Criar nomes de variáveis claros e significativos é uma prática essencial na programação. Aqui estão algumas regras e dicas para nomear variáveis em C:

1

Comece com uma letra ou sublinhado (`_`)

Nomes de variáveis devem começar com uma letra (a-z, A-Z) ou um sublinhado (`_`). Eles não podem começar com um número.

```
int idade;    // Correto
int altura;   // Correto
int 1salario; // Incorreto
```

2

Em seguida, use apenas letras, números e sublinhados.

Depois do primeiro caractere, você pode usar letras, números e sublinhados.

```
int altura2;    // Correto
int salario_3;  // Correto
```

3 Case-sensitive

Nomes de variáveis são sensíveis a maiúsculas e minúsculas. Isso significa que **idade** e **Idade** são variáveis diferentes.

```
int idade;  
int Idade;
```

4

Não use palavras reservadas

Evite usar palavras reservadas da linguagem C, como nomes de variáveis (por exemplo, int, float, return).

```
int int;    // Incorreto  
int return; // Incorreto
```

5

Escolha nomes significativos

Use nomes que façam sentido e descrevam o propósito da variável. Isso torna seu código mais legível e fácil de entender.

```
int idade;    // Correto  
int x;        // Correto, mas não descritivo
```

Inicialização de variáveis

Inicializar uma variável significa colocar um valor inicial dentro da caixa. Isso pode ser feito na mesma linha em que você declara a variável ou depois, no meio do seu código. Vamos ver alguns exemplos!

- `int idade = 25; // Declara "idade" e atribui o valor 25`
- `float altura; // Declara "altura" sem inicializar`
- `altura = 1.75; // Atribui o valor 1.75 a "altura" posteriormente`
- `double salario = 3000.50; // Declara "salario" e atribui o valor 3000.50`
- `char opcao = 'S'; // Declara "opcao" e atribui o caractere 'S'`

Aqui, estamos criando as variáveis e colocando valores iniciais nelas ao mesmo tempo.



Atenção

Para inicializar variáveis do tipo char você precisa colocar o caractere entre aspas simples ' '.

Inicializar variáveis é importante porque, se você tentar usar uma variável que não foi inicializada, ela pode conter lixo (valores aleatórios) que podem causar erros no seu programa.

Tipos de dados primitivos

São como etiquetas que você coloca nas suas caixas (variáveis) para indicar que tipo de valor elas podem armazenar. Diferentes tipos de dados são usados para diferentes propósitos. A Linguagem C tem 5 diferentes tipos básicos: char, int, float, void, double. Vamos explorar os tipos de dados primitivos em C.

Tipos de dados Inteiros (int)

Os inteiros são números sem casas decimais. Eles são usados quando você precisa contar ou trabalhar com números inteiros. Por exemplo, `int idade = 25;` armazena a idade de uma pessoa.

- `int idade = 25;`

Tipos de dados de ponto flutuante (float e double)

Os números de ponto flutuante são usados para representar números com casas decimais. Existem dois tipos principais: **float** e **double**. A diferença entre eles é a precisão. float ocupa menos espaço na memória, mas tem menos precisão do que double.

- `float altura = 1.75;`
- `double salario = 3000.50;`

Tipo de dados (void)

Void é um tipo de dado especial que representa a ausência de um tipo. Ele é usado em situações onde uma função não retorna nenhum valor ou quando um ponteiro não tem um tipo específico.

Tipos de dados – caractere (char)

O tipo char armazena caracteres como números inteiros que representam sua posição na tabela ASCII (American Standard Code for Information Interchange). A tabela ASCII mapeia caracteres (letras, números, símbolos) para números de 0 a 127. Por exemplo, o caractere 'A' é representado pelo número 65.

- char inicial = 'A';



Comentário

Quando usar float e quando usar double? Quando você está programando em C e precisa de números decimais com uma precisão muito alta e uma faixa de valores ampla, aí é hora de usar double. Essa é uma opção que oferece mais precisão do que o float, permitindo lidar com cálculos muito precisos, como em aplicações científicas ou financeiras. Então, se você está trabalhando com números que exigem muitos dígitos depois da vírgula e precisa de um intervalo maior de valores, o double é a escolha certa para garantir que suas operações sejam precisas e eficazes.

Uso de strings

Além de armazenar caracteres individuais, podemos também trabalhar com sequências de caracteres, conhecidas como strings que são sequências de caracteres armazenadas como *arrays* (conjuntos) de char, terminadas por um caractere especial chamado *caractere nulo* (\0). Esse caractere indica o fim da string.

- char nome[20] = "Alice"; // Declara um array de 20 caracteres e inicializa com "Alice"

No exemplo acima, nome pode armazenar até 19 caracteres mais o caractere nulo. A manipulação de strings requer cuidado para evitar *buffer overflows* (escrever além dos limites do array). Arrays e manipulação de strings serão abordados com mais detalhes posteriormente na disciplina.

Entrada e saída de dados

Aprenda neste vídeo sobre as funções de entrada e saída de dados em C. Descubra como usar printf para mostrar informações na tela e scanf para ler dados do usuário. Veja exemplos práticos e entenda como formatar e manipular dados de entrada e saída, essencial para tornar seus programas interativos e funcionais.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Entrada e saída de dados com C

Agora que você já entende variáveis e tipos de dados, é hora de aprender como obter informações do usuário e como mostrar resultados. Em programação, isso é conhecido como entrada e saída de dados. Vamos explorar as funções printf e scanf, que são fundamentais em C para lidar com a entrada e saída de dados.

Saída de dados com printf

A função `printf` é usada para exibir informações na tela. Pense em `printf` como uma maneira de pegar informações das suas "caixas" (variáveis) e mostrá-las ao mundo exterior.

Sintaxe básica de `printf`

A sintaxe básica de `printf` é:

```
printf("texto com formatação", variavel1, variavel2, ...);
```

Aqui, "texto com formatação" é uma string que pode conter texto e especificadores de formato para as variáveis que você deseja exibir.

Exemplo simples de `printf`: vamos começar com um exemplo simples, no qual exibimos uma mensagem na tela sem usar variáveis.



Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, `printf("Olá, Mundo!\n");` imprime a mensagem "Olá, Mundo!" na tela. O `\n` no final adiciona uma nova linha, movendo o cursor para a linha seguinte.

Exemplo de uso com variáveis: ao utilizar o `printf` para imprimir variáveis, é necessário colocar o nome da variável e o especificador de formato. Veja a sintaxe a seguir.

```
printf("%formato1 %formato2", variavel1, variavel2);
```

Aqui, "%formato1 %formato2" são os especificadores de formato correspondentes aos tipos das variáveis que você deseja exibir. Esses especificadores são elementos essenciais para controlar a formatação dos dados nas funções de entrada e saída da linguagem C, como `printf` e `scanf`. Cada especificador de formato é precedido por um caractere `%` e indica o tipo de dado da variável que será exibida. Para cada variável que você deseja imprimir, é necessário acrescentar um especificador de formato correspondente.

A seguir, você tem uma lista dos principais especificadores utilizados:

- **%d**: Imprime um inteiro no formato decimal.
- **%i**: Equivalente a **%d**.
- **%f**: Imprime um número de ponto flutuante no formato padrão.

- **%e**: Imprime um número de ponto flutuante na notação científica.
- **%c**: Imprime um único caractere.
- **%s**: Imprime uma cadeia (string) de caracteres.



Dica

Pesquise outros especificadores de formato e seus modificadores, que permitem personalizar a exibição dos dados.

Vamos ver um exemplo que exibe diferentes tipos de dados:



Conteúdo interativo

esse a versão digital para executar o código.

Nesse exemplo, você pôde ver a variável `idade` que é um inteiro que armazena o valor 25. A linha `printf("Idade: %d anos\n", idade)` imprime a saída: "Idade: 25 anos". Nesse caso, o `%d` é utilizado para formatar e exibir esse valor como um inteiro.

A variável `altura` é um float que armazena o valor 1.75 e a linha `printf("Altura: %.2f metros\n", altura)` vai imprimir a saída "Altura: 1.75 metros". Perceba que o especificador de formato utilizado é o `%.2f` que formata e exibe esse valor como um número de ponto flutuante com duas casas decimais.

Na linha seguinte, você pode ver a impressão de uma variável do tipo `double`. A saída do comando será "Saldo Bancário: 12345.67 reais". Perceba que é utilizado o mesmo especificador de formato que foi empregado no float `"%.2f"`.

A variável `inicial` é um caractere (`char`) que armazena a letra 'A'. `printf("Inicial do Nome: %c\n", inicial)` que irá imprimir "Inicial do Nome: A"; usa `%c` para formatar e exibir esse valor como um caractere.

Entrada de dados com `scanf`

Existem várias formas de obter entrada de dados em um programa, mas, aqui, focaremos o uso da função `scanf`. Essa é uma das formas mais comuns e úteis de ler dados do usuário em programas escritos em C.



Comentário

A função `scanf` é usada para ler dados fornecidos pelo usuário e armazená-los em variáveis. Pense em `scanf` como uma maneira de pegar informações do mundo exterior e colocá-las dentro das "caixas" (variáveis) do seu programa.

Sintaxe básica de scanf

A forma geral do comando scanf é a seguinte:

```
scanf("formato", &variavel);
```

Aqui, "formato" especifica o tipo de dado que você espera que o usuário insira, e &variavel é o endereço da variável onde o dado será armazenado.



Atenção

Você precisa sempre colocar o caractere "&" antes de cada variável que será lida.

Vamos analisar um exemplo no qual solicitamos ao usuário que insira sua idade. Lemos essa entrada e, em seguida, exibimos a idade inserida na tela. Este exemplo utiliza as funções printf e scanf.



Conteúdo interativo

esse a versão digital para executar o código.

A função printf("Digite sua idade: "); é usada para imprimir texto na tela. A string "Digite sua idade: " é exibida exatamente como está escrita. Esse texto serve como uma instrução para o usuário saber o que deve fazer a seguir.

A função scanf("%d", &idade); é usada para ler a entrada do usuário. O especificador de formato %d indica que esperamos um valor do tipo int (inteiro). O símbolo & antes do nome da variável (idade) é necessário para passar o endereço da variável, permitindo que scanf armazene o valor inserido pelo usuário diretamente nessa variável.

A função printf("Sua idade é: %d\n", idade); é usada novamente para imprimir texto na tela. Dessa vez, a string de formatação "Sua idade é: %d\n" inclui o especificador %d, que será substituído pelo valor da variável idade. O \n no final da string adiciona uma nova linha após a exibição da mensagem, movendo o cursor para a linha seguinte.

Entrada de Dados com scanf (Lendo Strings)

A função scanf também pode ser usada para ler strings do usuário, mas com uma importante ressalva: por padrão, ela lê apenas até o primeiro espaço em branco. Isso significa que se o usuário digitar um nome completo (com espaços), apenas a primeira parte do nome (até o primeiro espaço) será lida.

- char nome[50];
- printf("Digite seu nome: ");
- scanf("%s", nome);

- `printf("Nome digitado: %s\n", nome);`

Se o usuário digitar "João da Silva", apenas "João" será armazenado em `nome`. "da Silva" será deixado no buffer de entrada e poderá causar problemas em leituras subsequentes.

Lendo Strings com Espaços: `fgets`

Para ler strings com espaços em branco, a função `fgets` é a melhor opção. Ela lê uma linha inteira da entrada, incluindo espaços, até encontrar um caractere de nova linha (`\n`) ou atingir o tamanho máximo especificado.

- `char nome[50];`
- `printf("Digite seu nome completo: ");`
- `fgets(nome, 50, stdin);` // Lê no máximo 49 caracteres da entrada padrão (`stdin`)
- `// fgets inclui o '\n' na string, então podemos removê-lo se necessário`
- `nome[strcspn(nome, "\n")] = 0;`
- `printf("Nome completo digitado: %s\n", nome);`
- `fgets` recebe três argumentos:
 1. O array onde a string será armazenada.
 2. O tamanho máximo da string (incluindo o `\n` e o `\0`).
 3. O fluxo de entrada (geralmente `stdin` para a entrada padrão do teclado).

A linha `nome[strcspn(nome, "\n")] = 0;` remove o caractere de nova linha (`\n`) que `fgets` pode incluir na string. `strcspn` retorna o índice da primeira ocorrência de `\n` na string `nome`.



Resumindo

Dominar a entrada e saída de dados é essencial para interagir com os usuários e obter informações úteis para o seu programa. Pratique bastante usando `scanf` e `printf` para se familiarizar com essas funções.

Hora de codar

Agora você aplicará os conceitos estudados no nível básico, criando um programa em C que gerencia o cadastro de alunos em uma turma. Utilizaremos variáveis para armazenar informações como nome, idade e matrícula dos alunos. Operadores de atribuição serão usados para atualizar os dados conforme necessário.

As funções `printf` e `scanf` serão usadas para interagir com o usuário, permitindo a entrada dos dados dos alunos e a exibição das informações cadastradas. Essa prática reforçará seu entendimento sobre variáveis, tipos de dados e funções de entrada e saída, preparando-o para resolver problemas mais complexos.

Confira no vídeo como criar um programa em C para cadastrar os dados dos alunos. Aprenda sobre variáveis, operadores matemáticos e funções de entrada e saída. Siga o passo a passo para usar os comandos `printf` e `scanf`.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível novato

Criando as Cartas do Super Trunfo

Neste primeiro nível, seu objetivo é construir a base do nosso jogo Super Trunfo de Países: um sistema para cadastrar as cartas com informações sobre as cidades. Você vai praticar a leitura de dados do usuário, o armazenamento em variáveis e a exibição dessas informações na tela.

O que você vai fazer

Criar um programa em C que permita ao usuário inserir os dados de **duas** cartas do Super Trunfo. Para cada carta, o usuário deverá fornecer as seguintes informações:

- **Estado:** Uma letra de 'A' a 'H' (representando um dos oito estados). Tipo: `char`
- **Código da Carta:** A letra do estado seguida de um número de 01 a 04 (ex: A01, B03). Tipo: `char[]` (um array de caracteres, ou `string`)
- **Nome da Cidade:** O nome da cidade. Tipo: `char[]` (`string`)
- **População:** O número de habitantes da cidade. Tipo: `int`
- **Área (em km²):** A área da cidade em quilômetros quadrados. Tipo: `float`
- **PIB:** O Produto Interno Bruto da cidade. Tipo: `float`
- **Número de Pontos Turísticos:** A quantidade de pontos turísticos na cidade. Tipo: `int`

Após o usuário inserir os dados de cada carta, seu programa deve exibir na tela as informações cadastradas, de forma organizada e legível. Para cada carta, imprima cada informação em uma linha separada, com uma descrição clara. Por exemplo:

Carta 1:

Estado: A

Código: A01

Nome da Cidade: São Paulo

População: 12325000

Área: 1521.11 km²

PIB: 699.28 bilhões de reais

Número de Pontos Turísticos: 50

Carta 2:

Estado: B

Código: B02

Nome da Cidade: Rio de Janeiro

População: 6748000

Área: 1200.25 km²

PIB: 300.50 bilhões de reais

Número de Pontos Turísticos: 30

Requisitos funcionais

- O programa deve ler corretamente os dados de duas cartas do usuário via entrada padrão (teclado).
- O programa deve armazenar os dados lidos em variáveis apropriadas.
- O programa deve exibir os dados de cada carta na tela, formatados de forma clara e organizada, conforme o exemplo acima.

Requisitos não funcionais

- **Usabilidade:** O programa deve ser fácil de usar, com instruções claras para o usuário.
- **Legibilidade:** O código deve ser bem indentado, com comentários explicativos e nomes de variáveis significativos, facilitando a leitura e compreensão.
- **Corretude:** O programa deve funcionar corretamente, sem erros de compilação ou execução.

Simplificações para o desafio

- Você só precisa implementar o cadastro e a exibição de **duas** cartas.
- Neste nível, foque apenas na leitura, armazenamento e exibição das informações. Você **não precisa** implementar nenhuma lógica de comparação entre as cartas ou qualquer outro recurso adicional.
- **Não utilize** estruturas de repetição (como for ou while) ou estruturas de decisão (como if ou else). Seu código deve ser uma sequência simples de instruções.

Entregando seu Projeto

1. **Desenvolva seu projeto no GitHub:** Crie um repositório público no GitHub para o seu projeto. Recomendamos usar o GitHub Codespaces para facilitar o desenvolvimento.
2. **Crie o arquivo do seu código:** No seu repositório, crie um arquivo chamado `super_trunfo.c` com o seu código C.
3. **Compile e teste:** Compile e teste seu programa localmente para garantir que ele funciona corretamente.
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:** Copie o link do seu repositório no GitHub e envie-o através da plataforma SAVA, seguindo as instruções fornecidas.

Lembre-se: este é o primeiro passo. Concentre-se em entender os fundamentos de entrada, processamento e saída de dados em C. Nos próximos níveis, você adicionará mais funcionalidades e complexidade ao seu jogo.



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.

Conceitos de solução estruturada

Confira no vídeo dois conceitos de solução estruturada: modularidade e abstração. Esses conceitos permitem o desenvolvimento de programas mais organizados e eficientes.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Continuamos a utilizar a solução estruturada, essencial para escrever programas claros e organizados. Baseada nos conceitos de modularidade, abstração e um fluxo lógico consistente, a solução estruturada nos ajuda a resolver problemas complexos de maneira eficiente e organizada. Vamos nos aprofundar nesses conceitos para aprimorar nossa capacidade de solucionar problemas de forma eficaz.

Modularidade

Dividir um problema grande em subproblemas menores é uma técnica fundamental para a solução estruturada. Isso não só torna o problema mais manejável, mas também facilita a identificação e a correção de erros.

Imagine que você esteja desenvolvendo um sistema para calcular a média de temperaturas de uma semana. O problema pode ser dividido nas seguintes etapas. Observe!

Entrada de dados

Coletar as temperaturas diárias.

Saída de dados

Exibir a média das temperaturas.

Processamento de dados

Calcular a média das temperaturas.

A modularidade está diretamente ligada ao conceito de dividir um problema grande e complexo em problemas menores e mais manejáveis. Ela envolve escrever funções ou módulos que realizam tarefas específicas, tornando o código mais reutilizável e fácil de entender. Cada módulo deve ter uma única responsabilidade. Embora o conceito de função não seja o foco deste desafio, utilizaremos funções genéricas para ilustrar a modularidade.

Vamos usar o exemplo de cálculo da média de temperaturas para ilustrar a modularidade.

```

c

#include

// Função genérica para entrada de dados
void entradaDados() {
    // código para a função entradaDados
}

// Função genérica para cálculo da média
float calcularMedia() {
    // código para a função calcularMedia
}

// Função genérica para exibir dados
void exibirMedia() {
    // código para a função exibirMedia
}

int main() {

    entradaDados();
    media = calcularMedia();
    exibirMedia();

    return 0;
}

```

Nesse exemplo, temos três funções distintas para entrada, processamento e saída de dados, cada uma com uma responsabilidade clara. Lembre-se de que o conceito de função será abordado em detalhes em um nível posterior, mas é utilizado aqui para ilustrar a modularidade.



Comentário

Com a modularidade, é fácil alterar as funções de entrada, processamento e saída de dados sem afetar o restante do código. Por exemplo, se você quiser alterar a maneira como os dados são coletados (entrada de dados) ou como os resultados são exibidos (saída de dados), você só precisa modificar as funções correspondentes. Além disso, as funções de entrada e saída podem ser reutilizadas em outros programas ou contextos, aumentando a eficiência do desenvolvimento.

Abstração

Permite focar os aspectos mais importantes de um problema, ignorando os detalhes irrelevantes. Isso é importante para lidar com a complexidade.

Usando o exemplo do cálculo da média das temperaturas, em vez de se preocupar com os detalhes de como a entrada de dados é feita, você pode criar uma função `entradaDados` que abstrai esse processo. Assim, se precisar modificar a maneira como os dados são coletados, basta alterar essa função, mantendo o restante do código inalterado.

Ferramentas de planejamento de soluções

Assista ao vídeo e entenda como utilizar pseudocódigo e fluxogramas para planejar e representar soluções de programação. Veja exemplos práticos para facilitar a visualização e resolução de problemas complexos.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Uso de pseudocódigo e fluxogramas para problemas mais complexos

Para resolver problemas complexos de maneira eficiente, é essencial planejar a solução antes de começar a programar. Duas ferramentas eficazes para esse planejamento são o **pseudocódigo** e os **fluxogramas**. Essas ferramentas permitem visualizar a lógica do algoritmo, identificar possíveis problemas antecipadamente e comunicar a lógica para outras pessoas de maneira clara e concisa.

Pseudocódigo

É uma forma de descrever a lógica do algoritmo em linguagem natural, de maneira estruturada e fácil de entender. Ele não segue a sintaxe de uma linguagem de programação específica, mas permite que você organize suas ideias e planeje a estrutura do programa.

Veja um exemplo de pseudocódigo para verificar se um número é divisível por 2:

```
c
Início
  Solicitar ao usuário um número
  Receber o número fornecido
  Se o número for divisível por 2 então
    Exibir "O número é par"
  Senão
    Exibir "O número é ímpar"
Fim
```

Benefícios do pseudocódigo

Utilizar pseudocódigo oferece diversas vantagens para a estruturação de algoritmos e planejamento de soluções. Vamos conhecê-las!

Simplicidade

Como o pseudocódigo não segue a sintaxe rígida de uma linguagem de programação, ele é mais fácil de escrever e entender.

Clareza

Ajuda a descrever a lógica do programa de forma clara, permitindo que qualquer pessoa, independentemente do nível de conhecimento em programação, compreenda a solução proposta.

Foco na lógica

Ao usar pseudocódigo, você pode se concentrar na lógica do algoritmo sem se preocupar com detalhes de implementação.

Ferramentas para criação de pseudocódigo

Para criar e organizar pseudocódigos, você pode utilizar várias ferramentas on-line e off-line. Algumas opções populares incluem:

PSelnt

Software que ajuda a desenvolver algoritmos em pseudocódigo de forma visual, facilitando a compreensão e o aprendizado.

Portugol

Ferramenta que permite a escrita de pseudocódigos em português, proporcionando uma maneira simples e intuitiva de aprender lógica de programação.

Essas ferramentas oferecem recursos que facilitam a escrita, organização e visualização de pseudocódigos, tornando o processo de planejamento mais eficiente.

Fluxogramas

São representações gráficas de um algoritmo, usando símbolos padronizados para ilustrar os passos e o fluxo do processo. Eles são úteis para visualizar a lógica e a sequência das operações.

Antes de criar seu fluxograma, é importante conhecer os componentes principais, que estão descritos a seguir e representados na imagem.

1

Ação ou processo

Representado por um retângulo, indica uma operação ou conjunto de operações.

2

Decisão

Representado por um losango, indica um ponto onde uma decisão é feita, levando a diferentes caminhos.

3

Conector

Representado por um círculo, é usado para conectar diferentes partes do fluxograma.

4

Entradas e saídas de dados

Representado por um paralelogramo, indica pontos em que dados são recebidos ou enviados.

5 Início/Fim

Representado por um oval, indica o começo ou o fim do fluxo.

6

Seta de fluxo

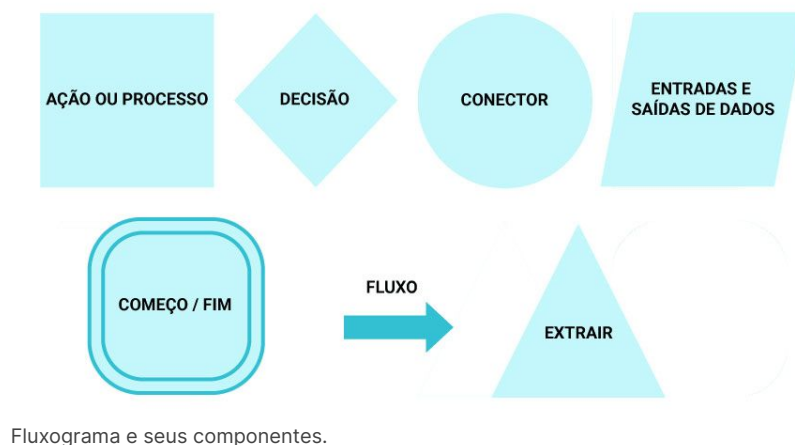
Representada por uma seta, indica a direção e a sequência do fluxo do processo.

7

Extrair

Representado por um triângulo, indica uma operação de extração de dados ou informações.

Observe agora o uso prático dos componentes citados.



Fluxograma e seus componentes.

Benefícios dos fluxogramas

Oferecem inúmeras vantagens para a visualização e compreensão de algoritmos, permitindo uma análise clara e organizada do fluxo do programa.

Visualização clara

Os fluxogramas permitem visualizar a sequência de operações e o fluxo lógico de um programa de forma clara e organizada.

Identificação de problemas

A representação gráfica ajuda a identificar erros lógicos e gargalos no processo antes mesmo de escrever uma linha de código.

Facilidade de compreensão

Ao usar símbolos gráficos padronizados, os fluxogramas tornam-se fáceis de entender, mesmo para aqueles que não têm conhecimento profundo em programação.

Exemplos práticos de representação de problemas

Vamos ver dois exemplos de algoritmos representados por fluxogramas, cada um acompanhado por uma imagem que ilustra os passos descritos.

Verificar se um número é divisível por 2

Esse algoritmo tem como objetivo verificar se um número fornecido pelo usuário é divisível por 2, ou seja, se o número é par. Isso é feito verificando se o resto da divisão do número por 2 é zero. Vamos ver como fica o passo a passo do fluxograma e a imagem que representa o processo:



- Início.
- Perguntar ao seu amigo o número.
- Receber a resposta.
- Calcular o resto da divisão do número por 2.
- Verificar se o resto é 0.
 - Se SIM, responder que o número é PAR.
 - Se NÃO, responder que o número é ÍMPAR.
- Fim.

Vejamos como fica o fluxograma!

Verificar se o número é um quadrado perfeito

Esse algoritmo tem como objetivo verificar se um número fornecido pelo usuário é um quadrado perfeito. Um número é considerado um quadrado perfeito se a sua raiz quadrada é um número inteiro. A seguir, estão os passos do fluxograma e a imagem que representa o processo:



- Início.
- Solicitar um número inteiro ao usuário.
- Receber a resposta do usuário com o valor N1.
- Calcular a raiz quadrada de N1.
- Verificar se a raiz quadrada de N1 é inteira.
 - Se SIM, exibir "O número informado é quadrado perfeito!"
 - Se NÃO, exibir "O número informado não é quadrado perfeito!"
- Fim.

Vejamos como fica o fluxograma!

Ferramentas para criação de fluxogramas

Para criar fluxogramas, você pode usar várias ferramentas on-line e off-line. Aqui estão algumas opções populares. Confira!

- **Bizagi**: permite criar fluxogramas e modelar processos de negócios.
- **Microsoft Word**: permite a criação de fluxogramas usando formas e conectores.

- **Microsoft PowerPoint:** facilita a criação de fluxogramas com uma interface intuitiva para desenhar formas e setas.

Essas ferramentas oferecem recursos que facilitam o desenho de fluxogramas detalhados e a visualização clara da lógica dos algoritmos.



Atenção

O uso de pseudocódigo e fluxogramas é fundamental para o planejamento de soluções de problemas complexos. Eles ajudam a visualizar a lógica do programa, identificar possíveis problemas antecipadamente e comunicar a lógica para outras pessoas de maneira clara e concisa. Utilizando essas ferramentas, você estará mais bem preparado para desenvolver programas eficientes e bem estruturados.

Operadores matemáticos

Assista ao vídeo para aprender sobre os operadores matemáticos em C, incluindo soma, subtração, multiplicação, divisão, operadores de atribuição e incremento/decremento. Veja como aplicar esses operadores em exemplos práticos e entenda seu funcionamento no código.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Operadores matemáticos

Vamos explorar agora os operadores matemáticos em C. Esses operadores são fundamentais para realizar cálculos e manipulações de dados em seus programas. Vamos abordar os operadores de soma, subtração, multiplicação e divisão, operadores de atribuição e operadores de incremento e decremento.

Operadores aritméticos

Os operadores aritméticos básicos em C são usados para realizar operações matemáticas simples: soma, subtração, multiplicação e divisão.

Operador soma (+): é usado para somar valores de duas ou mais variáveis. Exemplo:

```
int a = 5;  
int b = 3;  
int soma = a + b; // soma será 8
```


Operador subtração (-): é usado para calcular a diferença entre dois valores. Exemplo:

```
int a = 5;  
int b = 3;  
int subtracao = a - b; // subtracao será 2
```

Operador multiplicação (*): é usado para calcular o produto de dois valores, por exemplo:

```
int a = 5;  
int b = 3;  
int multiplicacao = a * b; // multiplicacao será 15
```

Operador divisão (/): é usado para calcular a razão de dois valores. Note que a divisão entre inteiros resulta em um inteiro, descartando a parte fracionária. Veja um exemplo:

```
int a = 10;  
int b = 3;  
int divisao = a / b; // divisao será 3
```

Operadores de atribuição

São usados para atribuir valores a variáveis. Eles podem ser combinados com operadores aritméticos para realizar uma operação e, em seguida, atribuir o resultado à variável.

Atribuição simples (=): é o operador de atribuição mais básico, usado para inicializar ou atualizar o valor de uma variável, como apresentado a seguir.

```
int a = 5; // a agora vale 5
```

Atribuição com soma (+=): adiciona um valor à variável e atribui o resultado à própria variável. Esse operador é usado para aumentar o valor de uma variável de forma eficiente. Veja!

```
int a = 5;  
a += 3; // a será 8 (equivalente a a = a + 3)
```

Atribuição com subtração (-=): subtrai um valor da variável e atribui o resultado à própria variável. Esse operador é usado para diminuir o valor de uma variável de forma eficiente. Veja!

```
int a = 5;  
a -= 3; // a será 2 (equivalente a a = a - 3)
```

Atribuição com multiplicação (*=): permite multiplicar uma variável por um valor e atribui o resultado à própria variável. Esse operador é usado para multiplicar o valor de uma variável de forma eficiente, como apresentado a seguir.

```
int a = 5;  
a *= 3; // a será 15 (equivalente a a = a * 3)
```

Atribuição com divisão (/=): realiza a divisão da variável por um valor especificado e atribui o resultado à própria variável, de forma eficiente. Aqui está um exemplo:

```
int a = 10;  
a /= 2; // a será 5 (equivalente a a = a / 2)
```

Operadores de incremento e decremento

São usados para aumentar ou diminuir o valor de uma variável em 1. Acompanhe!

Operador incremento (++): aumenta o valor de uma variável em 1. Esse operador é muito utilizado em loops e outras situações em que é necessário incrementar o valor de uma variável repetidamente. Por exemplo, no trecho de código a seguir a variável “i” é inicializada valendo 1 e na linha “i++” ela passa a valer 2.

```
int i =1;  
i++; // i terá o valor 2
```

O operador incremento pode ser utilizado de duas formas:

Pré-incremento

Incrementa a variável antes de seu uso. No exemplo a seguir a variável a é incrementada antes do seu valor ser atribuído à variável b. Ao final da operação, tanto a, como b terão o mesmo valor.

```
int a = 5;  
int b = ++a; // a será 6, b será 6
```

Poderíamos escrever o código anterior do seguinte modo:

```
int a = 5;  
int b;  
a++; // a incrementa o valor e vale 6  
b = a // b recebe o valor de a e assume o valor 6
```

Pós-incremento

Incrementa a variável após seu uso. No exemplo a seguir, primeiro é atribuído o valor de a a variável b e depois a é incrementada. Os valores de a e b, nesse caso, serão diferentes. Veja o código a seguir:

```
int a = 5;  
int b = a++; // a será 6, b será 5
```

Poderíamos escrever o código anterior da seguinte forma:

```
int a = 5;  
int b;  
b = a // b recebe o valor de a e assume o valor 5  
a++; // a incrementa o valor e vale 6
```

Operador decremento (--): diminui o valor de uma variável em 1. Esse operador é muito utilizado em loops e em outras situações em que é necessário diminuir o valor de uma variável repetidamente. Por exemplo, no trecho de código a seguir, a variável “i” é inicializada valendo 10 e na linha “i--” ela passa a valer 9.

```
int i = 10;  
i--; // i terá o valor 9
```

O operador decremento também pode ser utilizado de duas formas:

Pré-decremento

Decrementa a variável antes de seu uso, como apresentado no código a seguir:

```
int a = 5;  
int b = --a; // a será 4, b será 4
```

Pós-decremento

Decrementa a variável após seu uso, conforme a seguir:

```
int a = 5;  
int b = a--; // a será 4, b será 5
```

Praticando com operadores matemáticos

Vamos fazer um exercício prático para consolidar o aprendizado. Neste exemplo, vamos realizar algumas operações matemáticas em uma variável e mostrar os resultados.



Conteúdo interativo

esse a versão digital para executar o código.

Na função principal main, começamos declarando duas variáveis inteiras a e b, atribuindo-lhes os valores 10 e 5, respectivamente. Essas variáveis servirão como nossos operandos para as operações matemáticas.

A seguir, realizamos algumas operações aritméticas básicas. Primeiro, somamos a e b e armazenamos o resultado na variável soma. Em seguida, subtraímos b de a e armazenamos o resultado na variável subtração. Depois, multiplicamos a e b, armazenando o resultado em multiplicação. Finalmente, dividimos a por b e armazenamos o resultado inteiro em divisão.

Depois disso, utilizamos operadores de atribuição para modificar os valores de a e b. Acrescentamos 2 ao valor de a usando o operador `+=`, resultando em 12. Da mesma forma, multiplicamos b por 3 usando o operador `*=`, resultando em 15.

Em seguida, utilizamos os operadores de incremento e decremento para ajustar os valores de a e b. Incrementamos a em 1 usando `a++`, resultando em 13. Decrementamos b em 1 usando `b--`, resultando em 14.

Finalmente, exibimos todos os resultados usando a função `printf`. Mostramos os resultados das operações aritméticas e os novos valores de a e b após as operações de atribuição e incremento/decremento.



Recomendação

Dominar os operadores matemáticos é essencial para realizar cálculos e manipular dados em seus programas. Pratique bastante usando operadores aritméticos, de atribuição e de incremento e decremento para se familiarizar com essas funcionalidades.

Manipulação e conversão de tipos de dados

Aprenda neste vídeo como manipular variáveis inteiras e de ponto flutuante, além de entender a conversão entre tipos de dados em C. Veja exemplos práticos para garantir precisão e eficiência em seus programas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Manipulação de variáveis inteiras e de ponto flutuante

Vamos entender melhor como manipular variáveis inteiras e de ponto flutuante, além de aprender sobre a conversão entre tipos de dados. Essas habilidades são essenciais para lidar com cálculos mais complexos e garantir que seus programas funcionem corretamente ao trabalhar com diferentes tipos de informações. Vamos lá!

Variáveis inteiras

São usadas para armazenar números inteiros, ou seja, números sem parte decimal. Em C, o tipo de dado inteiro mais comum é `int`.

Veja a seguir um exemplo de manipulação de variáveis inteiras:

Para ilustrar a manipulação de variáveis inteiras, consideramos um programa simples no qual declaramos e inicializamos duas variáveis inteiras, a e b. O programa executa operações aritméticas básicas, como soma, subtração, multiplicação e divisão, e exibe os resultados no terminal.



Conteúdo interativo

esse a versão digital para executar o código.

Saída do programa

Mostra os resultados das operações aritméticas realizadas com as variáveis *a* e *b*. A soma de 10 e 3 resulta em 13, a diferença resulta em 7, o produto é 30 e a divisão inteira de 10 por 3 resulta em 3, pois a parte decimal é descartada.

```
Soma: 13
Diferença: 7
Produto: 30
Quociente: 3
```

Veja a seguir um exemplo de divisão de variáveis inteiras:

Para mostrar como a divisão de variáveis inteiras descarta a parte decimal, criamos um programa no qual dividimos 7 por 2.



Conteúdo interativo

esse a versão digital para executar o código.

Explicação da saída

A saída do programa mostra que a divisão inteira de 7 por 2 resulta em 3, pois a parte decimal é descartada. Esse exemplo ilustra como a divisão de inteiros em C sempre retorna um valor inteiro.

```
Quociente: 3
```

Variáveis de ponto flutuante

São usadas para armazenar números com parte decimal. Em C, os tipos de dados de ponto flutuante mais comuns são *float* e *double*.

Veja a seguir um exemplo de manipulação de variáveis de ponto flutuante:

Para demonstrar a manipulação de variáveis de ponto flutuante, consideramos um programa no qual realizamos operações aritméticas básicas com variáveis *float* e exibimos os resultados com duas casas decimais.



Conteúdo interativo

esse a versão digital para executar o código.

Explicação da saída

A saída do programa mostra os resultados das operações aritméticas realizadas com as variáveis `x` e `y`. A soma de 5.5 e 2.2 resulta em 7.70, a diferença é 3.30, o produto é 12.10 e a divisão de 5.5 por 2.2 resulta em 2.50. Os resultados são exibidos com duas casas decimais, demonstrando a precisão das operações com ponto flutuante.

```
Soma: 7.70
Diferença: 3.30
Produto: 12.10
Quociente: 2.50
```

Conversão entre tipos de dados

Às vezes, você precisa converter um tipo de dado em outro para realizar certas operações. Em C, isso pode ser feito de duas maneiras: conversão implícita e conversão explícita (casting).

Conversão implícita

Ocorre automaticamente quando você mistura diferentes tipos de dados em uma expressão. O compilador converte os tipos de dados para realizar a operação de acordo com o que o programador escreveu.

Veja a seguir um exemplo de conversão implícita: Para ilustrar a conversão implícita, consideramos um programa no qual uma variável inteira é somada a uma variável float, resultando em uma conversão automática para float.



Conteúdo interativo

esse a versão digital para executar o código.

Explicação da saída: a saída do programa mostra que a soma de `a` (10) e `b` (3.5) resulta em 13.50. A variável inteira `a` foi automaticamente convertida para float na expressão `a + b`, permitindo que o resultado seja um número de ponto flutuante. O que efetivamente ocorre neste processo é que o valor de `'a'` (10) é promovido internamente para 10.0 (um float) antes da soma ser realizada."

```
Resultado: 13.50
```

Riscos da conversão implícita: a conversão implícita pode ser útil, mas também apresenta riscos. Pode ocorrer perda de dados ou precisão quando tipos de dados de precisões diferentes são convertidos. Por exemplo, ao converter float para int, a parte decimal será perdida. Além disso, se você não estiver ciente de quando a conversão implícita acontece, isso pode levar a comportamentos inesperados em seu programa.

Conversão explícita (casting)

A conversão explícita é feita usando operadores de casting para forçar a conversão de um tipo de dado para outro.

Veja a seguir um exemplo de conversão explícita:

Para demonstrar a conversão explícita, criamos um programa em que uma variável inteira é convertida para float antes da divisão, garantindo um resultado de ponto flutuante.



Conteúdo interativo

esse a versão digital para executar o código.

Explicação da saída: a saída do programa mostra que a divisão de a (10) por b (3) resulta em 3.33. A variável a foi explicitamente convertida para float antes da divisão, garantindo que o resultado fosse um valor de ponto flutuante.

Quociente: 3.33

Para garantir que suas operações aritméticas sejam realizadas corretamente e evitar problemas de precisão e perda de dados, aqui estão algumas práticas recomendadas que você deve seguir. Confira!

Verifique os tipos

Sempre verifique os tipos de dados ao realizar operações aritméticas para evitar resultados inesperados.

Use casting quando necessário

Utilize casting explícito para garantir que as operações sejam realizadas no tipo de dado correto.

Cuidado com a precisão

Lembre-se de que variáveis de ponto flutuante podem introduzir erros de precisão. Use double para maior precisão quando necessário.

Hora de codar

Agora você aplicará os conceitos estudados no nível intermediário criando um programa em C que calcula a média de três notas inseridas pelo usuário. Utilizaremos variáveis para armazenar as notas, operadores matemáticos para somar e calcular a média, e operadores de atribuição para atualizar valores.

As funções printf e scanf serão usadas para interagir com o usuário, permitindo a entrada das notas e a exibição da média final. Esta prática reforçará seu entendimento sobre variáveis, tipos de dados e funções de entrada e saída, preparando-o para resolver problemas mais complexos. Assista ao vídeo e confira!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível aventureiro

Calculando Densidade Populacional e PIB per Capita

No nível básico, você criou um sistema para cadastrar as cartas do Super Trunfo. Agora, vamos adicionar mais detalhes e complexidade ao nosso jogo! Neste nível, você implementará a lógica para calcular e exibir duas novas propriedades importantes para cada cidade: a densidade populacional e o PIB per capita.

O que você vai fazer

Aprimore o programa em C que você criou no nível básico. O programa continuará lendo as mesmas informações do usuário (estado, código da carta, nome da cidade, população, área, PIB e número de pontos turísticos) para **duas** cartas. A diferença é que, agora, seu programa também deve:

1. **Calcular a Densidade Populacional:** Divida a população da cidade pela sua área. O resultado será a densidade populacional, que representa o número de habitantes por quilômetro quadrado. Armazene esse valor em uma variável do tipo float.
2. **Calcular o PIB per Capita:** Divida o PIB da cidade pela sua população. O resultado será o PIB per capita, que indica a riqueza média por pessoa na cidade. Armazene esse valor em uma variável do tipo float.
3. **Exibir os Resultados:** Além de exibir as informações lidas do usuário (como no nível básico), seu programa também deve exibir a densidade populacional e o PIB per capita calculados para cada cidade. Formate os valores de ponto flutuante com duas casas decimais.

Exemplo de Saída:

Carta 1:

Estado: A

Código: A01

Nome da Cidade: São Paulo

População: 12325000

Área: 1521.11 km²

PIB: 699.28 bilhões de reais

Número de Pontos Turísticos: 50

Densidade Populacional: 8102.47 hab/km²

PIB per Capita: 56724.32 reais

Carta 2:

Estado: B

Código: B02

Nome da Cidade: Rio de Janeiro

População: 6748000

Área: 1200.25 km²

PIB: 300.50 bilhões de reais

Número de Pontos Turísticos: 30

Densidade Populacional: 5622.24 hab/km²

PIB per Capita: 44532.91 reais

Requisitos funcionais

- Manter as funcionalidades do nível básico (leitura e exibição dos dados das cartas).
- Calcular e exibir corretamente a densidade populacional e o PIB per capita para cada cidade.

Requisitos não funcionais

- Manter os requisitos não funcionais do nível básico (usabilidade, legibilidade, corretude).
- Eficiência: O programa deve realizar os cálculos de forma eficiente.

Simplificações para o nível intermediário

- Você ainda só precisa lidar com **duas** cartas.
- Continue **sem usar** estruturas de repetição (for, while) ou estruturas de decisão (if, else).

Entregando seu Projeto

1. **Desenvolva seu projeto no GitHub:** Continue usando o mesmo repositório do GitHub que você criou no nível básico.
2. **Atualize o arquivo do seu código:** Atualize o arquivo `super_trunfo.c` com o seu código C, incluindo as novas funcionalidades.
3. **Compile e teste:** Compile e teste seu programa localmente para garantir que ele funciona corretamente, incluindo os cálculos da densidade populacional e PIB per capita.
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:** Copie o link do seu repositório no GitHub e envie-o através da plataforma SAVA. Certifique-se de que o link seja o mesmo do nível básico, para que possamos acompanhar seu progresso.

Agora você está pronto(a) para aplicar seus conhecimentos de operadores matemáticos em C! Lembre-se de documentar seu código com comentários claros e usar nomes de variáveis descritivos. No próximo nível, adicionaremos ainda mais recursos e desafios ao nosso jogo.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Modificadores de tipos de dados

Descubra neste vídeo como usar modificadores de tipo como unsigned e long para manipular grandes números e variáveis complexas em C. Aprenda a declarar, inicializar e comparar variáveis, garantindo precisão nos cálculos e desenvolvendo programas mais eficientes.



Conteúdo interativo
Acesse a versão digital para assistir ao vídeo.

Modificadores de tipos de dados

São usados para alterar as propriedades dos tipos de dados primitivos, permitindo uma manipulação mais precisa e eficiente das variáveis.

Os modificadores de tipos de dados permitem que você controle melhor como os dados são armazenados e manipulados no programa. Em C, os modificadores mais comuns são unsigned e long.

Veja a diferença a seguir.

Modificador unsigned

É usado para declarar variáveis que podem armazenar apenas valores positivos (incluindo zero). Ele pode ser aplicado aos tipos de dados int e char, dobrando a faixa positiva de valores que podem ser armazenados. Observe!

int

int pode armazenar valores que vão de -2,147,483,648 a 2,147,483,647, permitindo tanto números positivos quanto negativos. Já um unsigned int pode armazenar apenas valores positivos, dobrando a faixa positiva para 0 a 4,294,967,295.



char

char pode armazenar valores que vão de -128 a 127. Já um unsigned char pode armazenar apenas valores positivos, permitindo um intervalo de 0 a 255. Isso é útil para armazenar caracteres de conjuntos de caracteres estendidos ou para representar dados binários.

A tabela a seguir apresenta um resumo sobre os tipos de dados int e char.

Tipo	Intervalo de valores
int	-2,147,483,648 a 2,147,483,647
unsigned int	0 a 4,294,967,295
char	-128 a 127
unsigned char	0 a 255

Tabela comparativa: valores de int e char com e sem unsigned.
Sérgio Cardoso.


Quando usar unsigned? Imagine que você está desenvolvendo um jogo e precisa armazenar a pontuação do jogador. A pontuação nunca será negativa, então usar unsigned int permite armazenar pontuações maiores do que seria possível com um int comum. Outro exemplo seria ao trabalhar com pixels de uma imagem, que são representados por valores de 0 a 255; nesse caso, unsigned char é ideal.

Exemplo de unsigned int e int

Nesse exemplo, a variável signedNumber é declarada como int, e a variável unsignedNumber é declarada como unsigned int, ambas armazenando o valor 3000000000.

O especificador de formato %d é usado no printf para exibir valores int, enquanto %u é usado para exibir valores unsigned int.

Vamos executar esse programa e verificar como ficam as saídas.



Conteúdo interativo

esse a versão digital para executar o código.

A saída mostra que o valor 3000000000 é armazenado corretamente na variável unsigned int, mas excede o limite da variável int, resultando em um valor negativo devido ao overflow (situação em que um valor excede a capacidade de armazenamento de determinado tipo de dado). Isso ilustra a importância de escolher o tipo de dado correto para evitar problemas de armazenamento.

Número assinado: -1294967296

Número não assinado: 3000000000

Modificador long

É usado para aumentar a capacidade de armazenamento dos tipos de dados primitivos. Ele pode ser aplicado a int e double, permitindo armazenar valores maiores e com maior precisão.

Um int normal pode armazenar valores que vão de -2,147,483,648 a 2,147,483,647. Já um long int pode armazenar valores muito maiores, de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807, proporcionando uma faixa muito mais ampla para cálculos que exigem grandes números inteiros.

Um double pode armazenar valores com grande precisão, de ±1.7E-308 a ±1.7E+308. Um long double, por outro lado, oferece uma precisão ainda maior, permitindo armazenar valores de ±3.4E-4932 a ±1.1E+4932. Veja a tabela!

Tipo	Intervalo de valores
int	-2,147,483,648 a 2,147,483,647
long int	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
double	±1.7E-308 a ±1.7E+308
long double	±3.4E-4932 a ±1.1E+4932

Tabela Comparativa: Valores de int e double com e sem long.
Sérgio Cardoso.

Quando usar long? Se você precisa trabalhar com números que excedem a capacidade de um int normal (por exemplo, em cálculos astronômicos ou para armazenar a população mundial), long int ou long long int são necessários. Se a precisão de double não for suficiente para seus cálculos (em simulações físicas ou modelagem financeira, por exemplo), long double é a escolha adequada.

Exemplo de long int e int

Nesse exemplo, a variável regularNumber é declarada como int, enquanto bigNumber é declarada como long int. Ambos armazenam inicialmente o valor máximo de um int (2147483647). Em seguida, bigNumber é atualizado para 2147483648, que está fora do alcance de um int normal, mas dentro do alcance de um long int. Confira!



Conteúdo interativo

esse a versão digital para executar o código.

A saída mostra que tanto regularNumber quanto bigNumber podem armazenar o valor 2147483647. No entanto, quando bigNumber é atualizado para 2147483648, ele pode armazenar o valor corretamente, enquanto regularNumber não poderia armazenar esse valor por exceder o limite de um int.

```
Número regular (int): 2147483647
Número grande (long int): 2147483647
Número grande atualizado(long int): 2147483648
```

Veja a seguir um exemplo de long double e double:

Nesse exemplo, a variável preciseNumber é declarada como double, enquanto veryPreciseNumber é declarada como long double. Ambos armazenam o valor de π (pi), mas veryPreciseNumber armazena com uma precisão maior.



Conteúdo interativo

esse a versão digital para executar o código.

Número preciso (double): 3.141592653589793

Número muito preciso (long double): 3.141592653589793238463

A saída mostra que preciseNumber armazena o valor de π com uma precisão de 15 casas decimais, enquanto veryPreciseNumber armazena com uma precisão de 21 casas decimais, ilustrando a maior precisão de long double.

Veja a seguir um exemplo de uso combinado:

A variável largePositiveNumber é declarada como unsigned long int, permitindo armazenar um número positivo extremamente grande. O especificador de formato %lu é usado no printf para exibir valores unsigned long int.



Conteúdo interativo

esse a versão digital para executar o código.

A saída do programa é:

Número positivo grande: 4000000000

A saída mostra que a variável `largePositiveNumber`, declarada como `unsigned long int`, armazena corretamente um valor extremamente grande (4000000000). Utilizando o modificador `unsigned long`, a faixa de valores possíveis para armazenamento é ampliada, evitando problemas de overflow e garantindo que valores positivos muito grandes sejam armazenados e manipulados corretamente.

Prática recomendada

Para garantir que suas operações com tipos de dados modificados sejam realizadas corretamente e para evitar problemas de overflow ou perda de precisão, aqui estão algumas práticas recomendadas que você deve seguir. Acompanhe!

Escolha o tipo de dado apropriado

Utilize modificadores de tipo quando precisar armazenar valores fora do alcance dos tipos de dados primitivos padrão.

Verifique a compatibilidade de tipos

Sempre verifique a compatibilidade dos tipos de dados ao realizar operações aritméticas para evitar resultados inesperados.

Use especificadores de formato corretos

Utilize os especificadores de formato corretos ao exibir valores com `printf` para garantir que os dados sejam exibidos corretamente.

Operadores relacionais

Assista ao vídeo e aprenda a utilizar operadores relacionais (`>`, `<`, `==`, `!=`) para comparar variáveis e valores em C. Confira a importância das comparações e como elas são usadas para avaliar condições e tomar decisões em programas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Introdução aos operadores relacionais

Eles são fundamentais na programação, pois permitem a comparação entre variáveis. Usados para estabelecer relações entre valores, os operadores relacionais retornam valores booleanos (1 para verdadeiro e 0 para falso). Eles são essenciais para a tomada de decisões em programas, mas, aqui, vamos focar o uso dos operadores relacionais sem o emprego de estruturas de controle como `if` ou `while`.

Principais operadores relacionais

Os operadores relacionais disponíveis na linguagem C são:

- > (maior que)
- < (menor que)
- >= (maior ou igual a)
- <= (menor ou igual a)
- == (igual a)
- != (diferente de)

Esses operadores são usados para comparar dois operandos, retornando 1 se a comparação for verdadeira e 0 se for falsa.

Vejamos alguns exemplos de como esses operadores funcionam na prática!

Comparação simples

Neste exemplo, temos duas variáveis inteiras a e b. Usamos os operadores relacionais para comparar a e b, imprimindo os resultados das comparações. As variáveis a e b são comparadas utilizando os operadores >, <=, e !=.

```
c
#include

int main() {
    int a = 10;
    int b = 20;

    printf("a > b: %d\n", a > b);
    printf("a < b: %d\n", a < b);
    printf("a == b: %d\n", a == b);
    printf("a != b: %d\n", a != b);

    return 0;
}
```

A saída do programa será:

```
a > b: 0
a < b: 1
a==b: 0
a !=b: 1
```

Os resultados da execução do programa têm o seguinte significado:

- a > b: 0 indica que a não é maior que b.
- a < b: 1 indica que a é menor que b.
- a == b: 0 indica que a não é igual a b.

- `a != b`: 1 indica que `a` é diferente de `b`.

Comparação com tipos de dados diferentes

Vamos considerar uma comparação entre tipos de dados inteiros e de ponto flutuante.

Aqui temos uma variável inteira `x` e uma variável de ponto flutuante `y`. Comparamos `x` e `y` utilizando operadores relacionais. A variável `x` é comparada com `y` utilizando os operadores `==` e `!=`.

```
c
#include

int main() {
    int x = 5;
    float y = 5.0;

    printf("x == y: %d\n", x == y);
    printf("x != y: %d\n", x != y);

    return 0;
}
```

A saída do programa será

- `x == y`: 1
- `x != y`: 0

A saída `x == y`: 1 indica que `x` é igual a `y` após a conversão implícita de tipos. Já em `x != y`: 0 indica que `x` não é diferente de `y`.

Observação: Antes da comparação, o compilador realiza uma conversão implícita do tipo `int` (`x`) para `float`. Isso ocorre porque o operador relacional (`==` ou `!=`) precisa comparar valores do mesmo tipo. Nesse caso, o valor de `'x'` (5) é convertido para 5.0 (`float`) antes da comparação com `'y'`.

Comparação entre variáveis de tipos diferentes com conversão explícita

Às vezes, é necessário fazer conversões explícitas para garantir a comparação correta. Nesse exemplo, a variável `num` é convertida explicitamente para `float` antes da comparação com `result`.


```
c
#include

int main() {
    int num = 10;
    float result = 10.0;

    int comparison = (float)num == result;

    printf("num == result: %d\n", comparison);

    return 0;
}
```

A saída do programa será:

```
num == result: 1
```

A comparação entre num e result retorna 1, indicando que são iguais após a conversão explícita de num para float.

Os operadores relacionais são uma ferramenta poderosa para comparar variáveis e valores em C. Eles permitem que você avalie condições e tome decisões com base nos resultados dessas comparações.

Hora de codar

Você aplicará conceitos de modificadores de tipos de dados e operadores relacionais para desenvolver um sistema de gerenciamento de inventário. Você implementará funcionalidades que permitem comparações entre diferentes produtos, como quantidade em estoque e valor total, utilizando operadores relacionais sem estruturas de controle.

Além disso, aprenderá a manipular grandes números com precisão, usando modificadores de tipo como unsigned e long. Esses conhecimentos são fundamentais para desenvolver sistemas robustos e eficientes, capazes de lidar com dados variados e realizar cálculos complexos de forma precisa. Assista ao vídeo e confira!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Desafio: nível mestre

Batalha de Cartas no Super Trunfo

Prepare-se para o desafio final! Agora que você já sabe cadastrar cartas e calcular atributos importantes, é hora de colocar suas cartas para lutar! Neste nível avançado, você implementará a lógica de comparação entre duas cartas do Super Trunfo, determinando a vencedora de cada atributo e introduzindo o conceito de "Super Poder". Prepare-se para usar todo o seu conhecimento de operadores relacionais, modificadores de tipo e conversão de tipos em C!

O que você vai fazer

Seu programa em C deve agora permitir ao usuário inserir os dados de **duas** cartas, assim como nos níveis anteriores, mas com uma mudança importante: a população agora deve ser armazenada como unsigned long int para acomodar números maiores. As demais informações permanecem com os mesmos tipos. Após a leitura dos dados, seu programa deverá:

1. **Calcular a Densidade Populacional e o PIB per Capita:** Assim como no nível intermediário, calcule e armazene esses valores.
2. **Calcular o Super Poder:** Para cada carta, calcule o "Super Poder" somando todos os atributos numéricos (população, área, PIB, número de pontos turísticos, PIB per capita e o *inverso* da densidade populacional – quanto menor a densidade, maior o "poder"). Armazene o Super Poder como float.
Atenção: Preste muita atenção na conversão de tipos ao somar valores de tipos diferentes!
3. **Comparar as Cartas:** Compare as duas cartas atributo por atributo (exceto estado, código e nome), incluindo o Super Poder. Lembre-se: para a densidade populacional, a carta com o *menor* valor vence; para os demais atributos (incluindo Super Poder), a carta com o *maior* valor vence.
4. **Exibir os Resultados das Comparações:** Para cada atributo, imprima na tela qual carta venceu (Carta 1 ou Carta 2), exibindo o resultado da comparação (1 para verdadeiro – Carta 1 vence – e 0 para falso – Carta 2 vence).

Exemplo de Saída (após a entrada dos dados):

Comparação de Cartas:

População: Carta 1 venceu (1)

Área: Carta 1 venceu (1)

PIB: Carta 1 venceu (1)

Pontos Turísticos: Carta 1 venceu (1)

Densidade Populacional: Carta 2 venceu (0)

PIB per Capita: Carta 1 venceu (1)

Super Poder: Carta 1 venceu (1)

Requisitos funcionais

- Manter as funcionalidades dos níveis básico e intermediário (leitura, cálculo e exibição dos dados das cartas, incluindo densidade populacional e PIB per capita).
- Calcular corretamente o Super Poder para cada carta.
- Comparar corretamente as cartas atributo por atributo, considerando a regra específica para densidade populacional.
- Exibir os resultados das comparações de forma clara e organizada.

Requisitos não funcionais

- Manter os requisitos não funcionais dos níveis anteriores (usabilidade, legibilidade, corretude, eficiência).

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.

Parabéns por chegar ao nível avançado! Este desafio integra os conceitos que você aprendeu até agora. Demonstre suas habilidades em C e crie um sistema de comparação de cartas robusto e eficiente!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Considerações finais

Você demonstrou dedicação e habilidade ao aplicar os conceitos estudados ao longo deste conteúdo. Aprendeu sobre a importância de variáveis, tipos de dados, operadores matemáticos e de atribuição, bem como a utilização das funções de entrada e saída em C. Esses fundamentos são essenciais para qualquer programador e são a base para resolver problemas de forma estruturada e eficiente.

Ao longo dessa atividade, você:

- **Utilizou variáveis e tipos de dados** : aprendeu a declarar, inicializar e manipular variáveis de diferentes tipos, como int, float, e char.
- **Explorou operadores matemáticos e de atribuição** : aplicou operadores para realizar cálculos e atualizar valores, entendendo como são usados em diferentes contextos.
- **Interagiu com o usuário** : utilizou as funções printf e scanf para exibir mensagens e receber entradas, criando uma interface de linha de comando interativa.
- **Organizou e estruturou código** : praticou a escrita de código claro e bem organizado, facilitando a leitura e a manutenção.

Na sua prática, é sempre importante lembrar:

- A **prática constante** é essencial para solidificar os conceitos aprendidos. Experimente resolver diferentes tipos de problemas e desafios.
- Sempre que possível, **revise seu código** e procure maneiras de torná-lo mais eficiente e legível. A **refatoração** é uma habilidade valiosa.
- Mantenha-se curioso e continue **aprendendo**. Explore outras estruturas de dados, algoritmos e paradigmas de programação.
- Engaje-se em **comunidades de programação**, como fóruns e grupos de estudo. Compartilhar conhecimentos e dúvidas pode acelerar seu aprendizado.
- Não tenha medo de enfrentar problemas mais complexos. Cada novo **desafio** é uma oportunidade de crescimento.

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.