**Explicando um pouco mais sobre passagem de argumentos por valor e por referência**

**Em relação as duas aulas anteriores leia abaixo um pouco mais sobre a Passagem de argumentos para função por valor e por referência**

Quando você chama uma função você envia para ela argumentos, valores que serão usados por esta função.

Então você pode enviar o que desejar, desde que esteja de acordo com o esperado pela função. Ou seja, de acordo com os parâmetros que a função espera receber.

No caso do código da aula anterior  a função void TrocaNumeros(int Num1, int Num2) espera receber dois valores inteiros e que serão armazenados nas variáveis locais a esta função Num1e Num2.

Logo quando você chama a função assim TrocaNumeros(Num1, Num2); isso indica que os valores contidos nas variáveis Num1 e Num2 que pertencem a função main, irão para a função TrocaNumeros e serão colocados dentro dos parâmetros desta função e que por mera concidência possuem o mesmo nome Num1 e Num2.

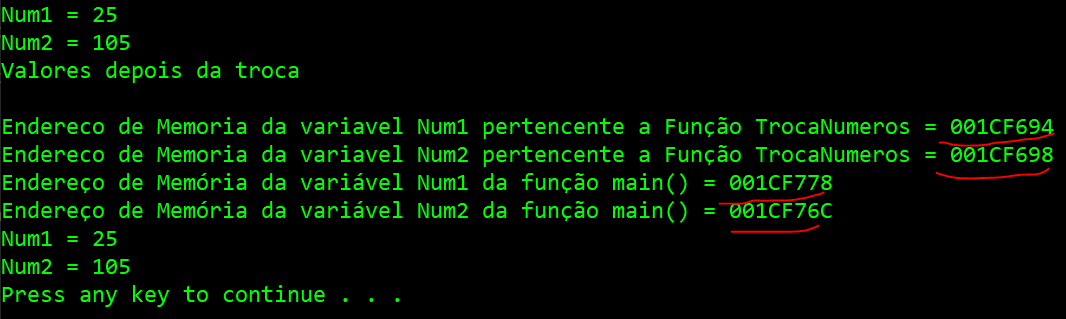
Lembrando que apesar dos nomes serem os mesmos as variáveis Num1 e Num2 declaradas na função main são diferentes das variáveis Num1 e Num2 criadas apenas durante a execução da função. Ou seja, estas variáveis terão endereços de memória diferentes e portanto para o computador(CPU) elas serão diferentes.

Então você chamou a função assim: **TrocaNumeros(Num1, Num2);**

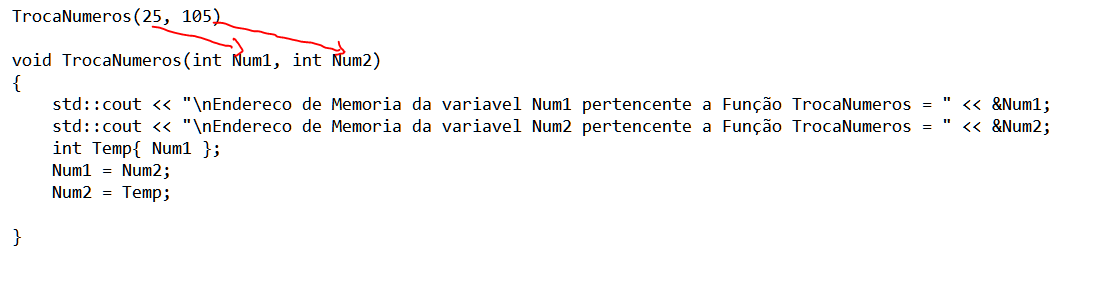
Vou usar este código da aula para explicar a passagem de parâmetros por valor

1. #include <iostream>
3. void TrocaNumeros(int Num1, int Num2);
5. int main()
6. {
7. setlocale(LC\_ALL, "portuguese");
8. int Num1, Num2;
9. std::cout << "\nDigite o primeiro Número: ";
10. std::cin >> Num1;
11. std::cout << "\nDigite o Segundo Número: ";
12. std::cin >> Num2;
13. std::cout << "\nValores antes da troca\n";
14. std::cout << "\nNum1 = " << Num1;
15. std::cout << "\nNum2 = " << Num2;
16. std::cout << "\nValores depois da troca\n";
18. TrocaNumeros(Num1, Num2);
19. std::cout << "\nEndereço de Memória da variável Num1 da função Main() = " << &Num1;
20. std::cout << "\nEndereço de Memória da variável Num1 da função Main() = " << &Num2;
22. std::cout << "\nNum1 = " << Num1;
23. std::cout << "\nNum2 = " << Num2 << "\n";
25. system("PAUSE");
26. return 0;
27. }
29. void TrocaNumeros(int Num1, int Num2)
30. {
31. std::cout << "\nEndereco de Memoria da variavel Num1 pertencente a Função TrocaNumeros = " << &Num1;
32. std::cout << "\nEndereco de Memoria da variavel Num2 pertencente a Função TrocaNumeros = " << &Num2;
33. int Temp{ Num1 };
34. Num1 = Num2;
35. Num2 = Temp;
37. }

Como exemplo digamos que o usuário digitou 25 e 105 na entrada



Veja que ao chamar a função será o mesmo que chamar assim TrocaNumeros(25, 105) ou seja, quando o compilador ver este comando TrocaNumeros(Num1, Num2) ele primeiro vai pegar o valor contido em Num1 e o valor contido em Num2 e depois envia estes valores para a função



Então a função TrocaNumeros está recebendo apenas o valor das variáveis Num1 e Num2 declaradas na função main( )

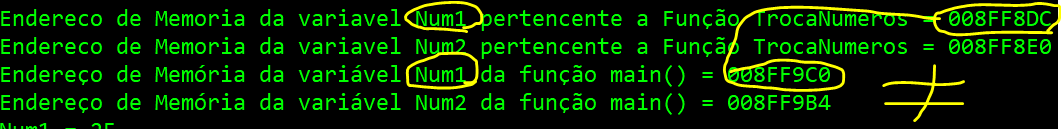
Logo o trecho de código da função TrocaNumeros vai apenas mudar os valores das variáveis locais a esta função e que por coincidência tem o mesmo nome mas são diferentes das variáveis Num1 e Num2 declaradas na função main().

Logo

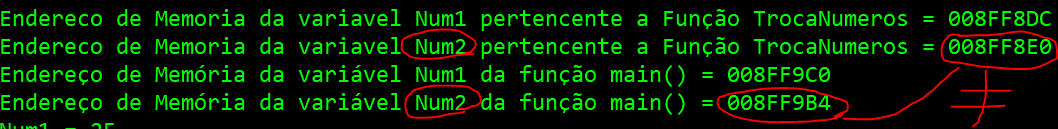
1. int Temp{ Num1 };  // Temp passa a ter o valor 25
2. Num1 = Num2;  // Num1 passa a ter o valor 105
3. Num2 = Temp; // Num2 passar a ter o valor 25

Contudo o que está sendo modificado são apenas os valores das variáveis locais Num1 e Num2 da função TrocaNumeros

Observe que as variáveis Num1 de main() e Num1 da função TrocaNumeros possuem endereços de memórias diferentes



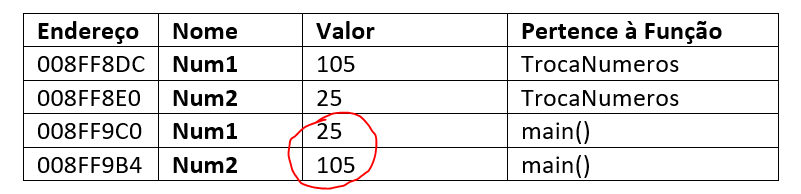
Observe que as variáveis Num2 de main() e Num2 da função TrocaNumeros também possuem endereços de memórias diferentes e assim são variáveis que possuem locais de armazenamento diferentes, mas tem nomes iguais. Porém para o computador o que importa são os endereços de memória das variáveis, pois é com eles que ele conseguir ler, alterar valores destas variáveis.



Logo quando a função está fazendo

1. int Temp{ Num1 };  // Temp passa a ter o valor 25
2. Num1 = Num2;  // Num1 passa a ter o valor 105
3. Num2 = Temp; // Num2 passar a ter o valor 25

A memória estará assim quando chegarmos ao final da função antes dele encerrar



Observe que os valores que são trocados ocorrem apenas nas variáveis da função TrocaNumeros, as variáveis da função main não são trocadas pois a função TrocaNumeros está manipulando apenas suas próprias variáveis locais Num1 e Num2. Ou seja, a função TrocaNumeros não tem como modificar, acessar as variáveis Num1 e Num2 da função main.

**E ENTÃO SURGE A PERGUNTA COMO FAZER COM QUE A FUNÇÃO TrocaNumeros consiga alterar os valores das variáveis Num1 e Num2 da função main?**

É ai que entram os ponteiros ou as referências

Vamos ver com os ponteiros

Veja que neste caso você irá indicar ao compilador que não vai mais enviar o VALOR das variáveis e sim você agora vai enviar para a função TrocaNumeros os endereços destas variáveis Num1 e Num2 de main

1. TrocaNumeros(&Num1, &Num2);

Veja que com & na frente você agora quer que o compilador pegue o valor do endereço de memória das variáveis Num1 e Num2 e envie para a função TrocaNumeros

Mas ai surge uma questão, pois se você mantiver a estrutura void TrocaNumeros(int Num1, int Num2) da função TrocaNumeros dará erro , pois neste caso esta assinatura(estrutura) da função espera receber dois números inteiros. Ela não está esperando receber dois endereços de memória. Logo precisamos adequar a função TrocaNumeros para que ela consiga receber em suas variáveis valores do tipo endereço de memória..

E qual é a variável capaz de armazenar endereços de memória?

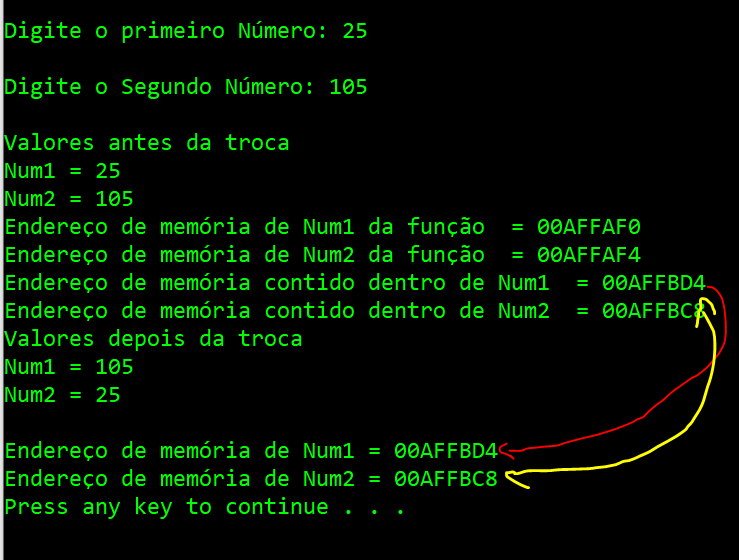
Isso mesmo! Ponteiros!

Logo basta colocar a função como sendo capaz de receber valores de endereço.

Ficará assim: void TrocaNumeros(int\* Num1, int\* Num2)

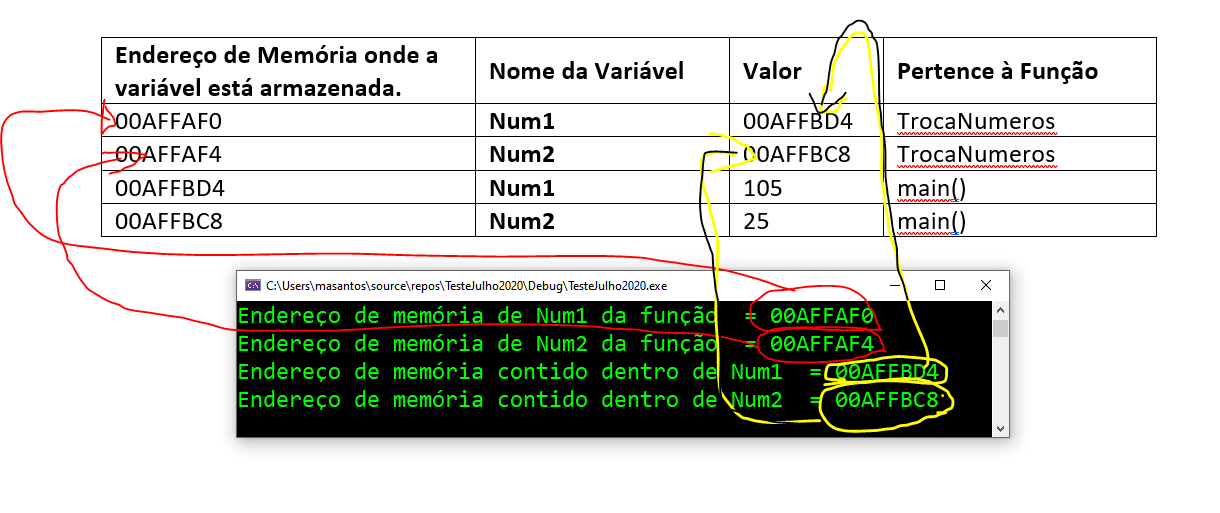
Então no exemplo do código abaixo ao chamarmos a função assim: TrocaNumeros(&Num1, &Num2);

O compilador vai buscar os valores dos endereços de Num1 e Num2 de main e vai chamar assim:  TrocaNumeros(00AFFBD4, 00AFFBC8);

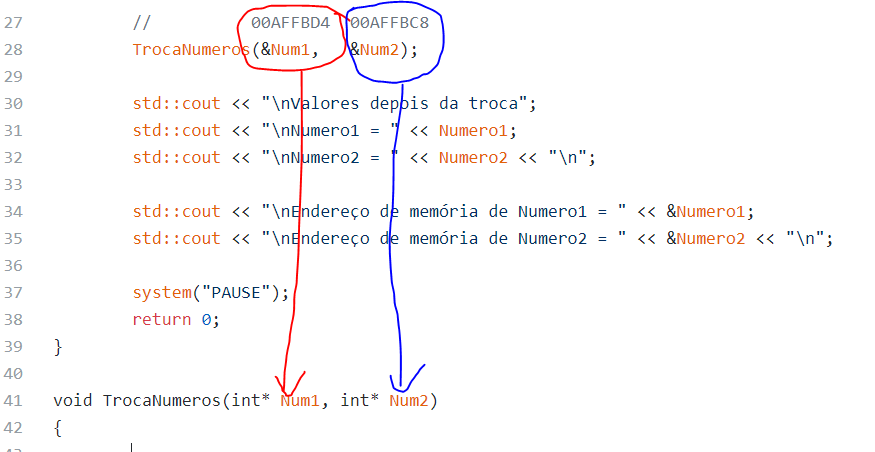


Chamamos isso de PASSAGEM DE ARGUMENTOS PARA A FUNÇÃO POR REFERÊNCIA! SÓ QUE AQUI USANDO PONTEIROS!

Veja como estará a memória depois da função ser chamada e antes dela ser encerrada



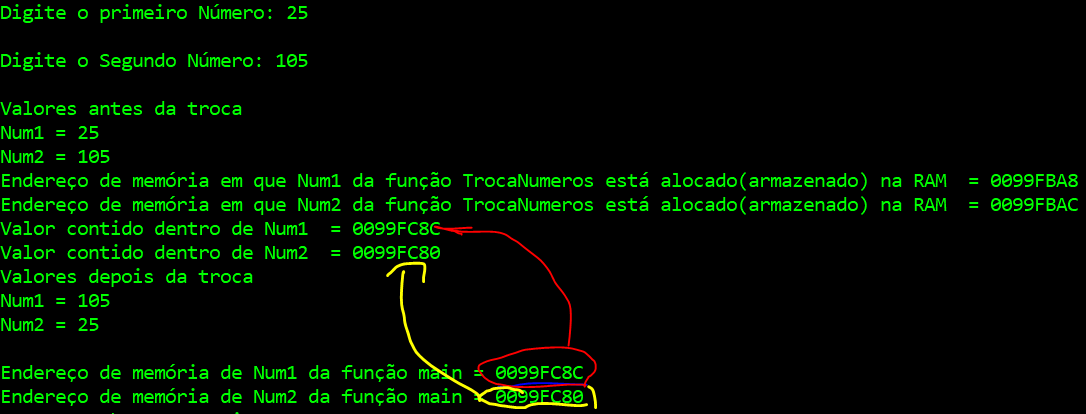
Observe que as variáveis locais Num1 e Num2 pertencentes a função TrocaNumeros estão alocadas(armazenadas) em um endereço da RAM. Porém como elas são variáveis ponteiros elas possuem dentro delas a informação de um endereço de memória. Foi esta informação que foi passada para a função



E é isso que vai permitir a função TrocaNumeros Trocar os valores de Num1 e Num2 da função main, pois agora a função recebeu os endereços destas variáveis e com isso temos o código a seguir(*mudei os nomes das variaveis para ficarem "iguais" pois muitos exemplos em livros e tutoriais é feito desta forma e normalmente confuder sua cabeça no início*.)

1. #include <iostream>
3. void TrocaNumeros(int\* Num1, int\* Num2);
5. int main()
6. {
7. setlocale(LC\_ALL, "portuguese");
8. int Num1, Num2;
9. std::cout << "\nDigite o primeiro Número: ";
10. std::cin >> Num1;
11. std::cout << "\nDigite o Segundo Número: ";
12. std::cin >> Num2;
13. std::cout << "\nValores antes da troca";
14. std::cout << "\nNum1 = " << Num1;
15. std::cout << "\nNum2 = " << Num2;
17. TrocaNumeros(&Num1, &Num2);
19. std::cout << "\nValores depois da troca";
20. std::cout << "\nNum1 = " << Num1;
21. std::cout << "\nNum2 = " << Num2 << "\n";
23. std::cout << "\nEndereço de memória de Num1 da função main = " << &Num1;
24. std::cout << "\nEndereço de memória de Num2 da função main = " << &Num2 << "\n";
26. system("PAUSE");
27. return 0;
28. }
30. void TrocaNumeros(int\* Num1, int\* Num2)
31. {
33. std::cout << "\nEndereço de memória em que Num1 da função TrocaNumeros está alocado(armazenado) na RAM = " << &Num1;
34. std::cout << "\nEndereço de memória em que Num2 da função TrocaNumeros está alocado(armazenado) na RAM = " << &Num2;
35. std::cout << "\nValor contido dentro de Num1 = " << Num1;
36. std::cout << "\nValor contido dentro de Num2 = " << Num2;
37. int Temp{ \*Num1 };
38. \*Num1 = \*Num2;
39. \*Num2 = Temp;
41. }

Ao executar novamente o código os endereços irão mudar pois a memória RAM é dinâmica e aloca espaço de acordo com a disponibilidade. Logo ao executar as variáveis vão ser alocadas em endereços de memória diferentes da execução anterior



O que significa então estes comandos abaixo da função TrocaNumeros?

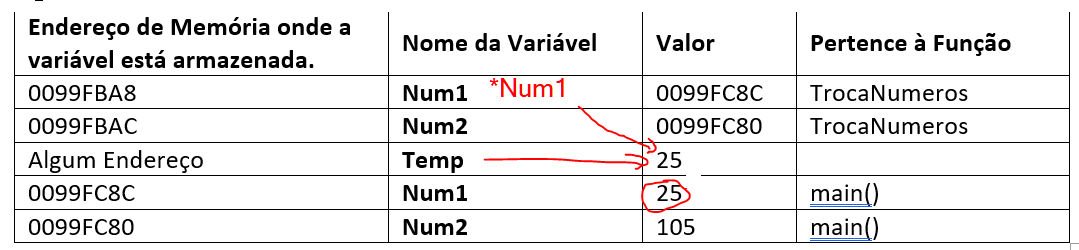
1. int Temp{ \*Num1 };
2. \*Num1 = \*Num2;
3. \*Num2 = Temp;

\*Num1 significa o desreferenciamento do ponteiro Num1. Logo é para ser buscado o valor contido no endereço de memória que Num1 está apontando.

Num1 da função TrocaNumeros contém dentro dele o endereço 009FC8C que é o endereço da variável Num1 da função main

Logo \*Num1 é o mesmo que \*009FC8C Ou seja, qual o valor que está armazenado no endereço de memória 009DC8C?

Veja a tabela de memória para esta execução antes da troca e quando a função está ainda no comando int Temp {\*Num1}



Então \*Num1 é 25 e portantoi Temp recebe 25

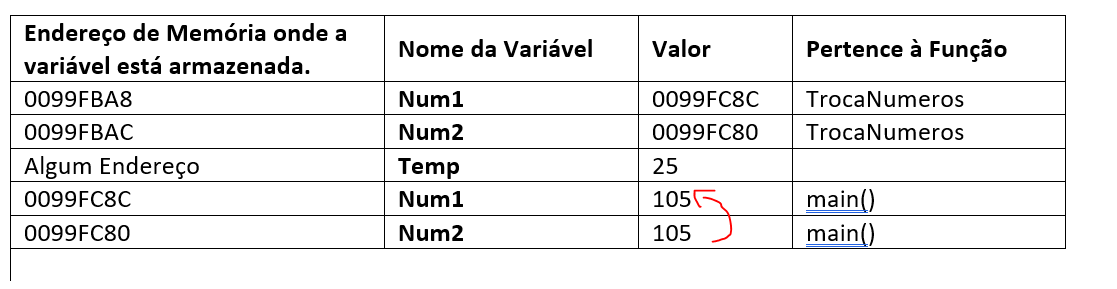
Depois temos o comando

1. \*Num1 = \*Num2;

Ou seja, trata-se de um simples comando de atribuição. O comando quer que o que está em \*Num2 vá para dentro de \*Num1

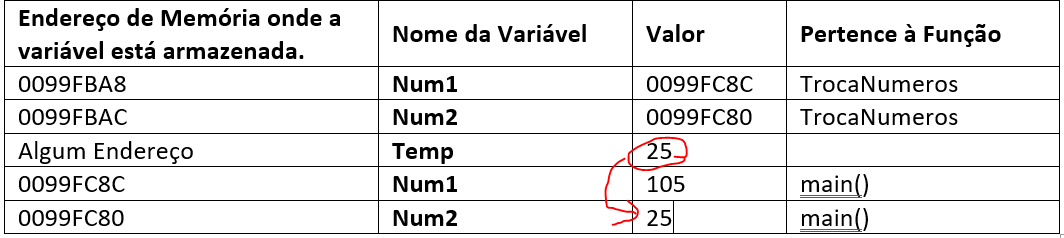
Traduzindo Pego o valor da variável apontada por Num1 e coloque este valor como sendo o novo valor da variável apontada por Num2.

Ficaremos assim:

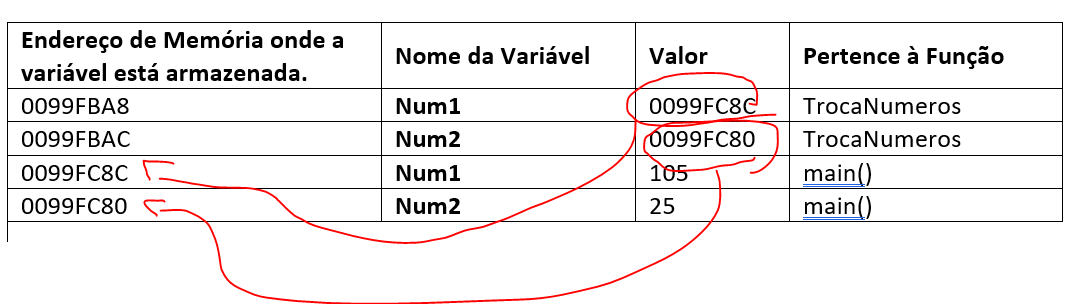


Agora o comando \*Num2 = Temp;

Ou seja, o valor contido em Temp vai para a variável apontada por Num2. Assim ficamos desta forma



E é por isso que teremos a troca, pois agora estamos manipulando o endereço de memória das variáveis e não apenas seu valor!



Com Referências o mecanismo é bem parecido que a dos ponteiros e a diferença são as que já expliquei nas aulas

Abraço

**Conteúdo do curso**

Seção 1: Introdução e Instalação do Ambiente de Programação

17 / 18 | 35m17 de 18 aulas concluídas35m

Seção 2: Primeira Lista de Exercícios e Resolução desta Primeira Lista de Exercícios

7 / 7 | 12m7 de 7 aulas concluídas12m

Seção 3: Conceitos Importantes Sobre Programação de Computadores

7 / 7 | 55m7 de 7 aulas concluídas55m

Seção 4: Criando e Manipulando Variáveis

22 / 22 | 1h 36m22 de 22 aulas concluídas1h 36m

Seção 5: Entrada de Dados e Programando com Operadores Aritméticos, Relacionais e Lógicos

18 / 18 | 1h 45m18 de 18 aulas concluídas1h 45m

Seção 6: Estruturas de Decisão Condicional (if-else |Operador Ternário ? | Switch Case)

8 / 8 | 46m8 de 8 aulas concluídas46m

Seção 7: Estruturas de Repetição(Loops): while, do-while, for

10 / 10 | 47m10 de 10 aulas concluídas47m

Seção 8: Funções Conceitos e Desenvolvimento

25 / 25 | 2h 28m25 de 25 aulas concluídas2h 28m

Seção 9: \*Ponteiros: conceitos e prática

10 / 10 | 1h 23m10 de 10 aulas concluídas1h 23m

Seção 10: &Referências: Conceitos e Prática

7 / 8 | 52m7 de 8 aulas concluídas52m

* Aula concluída. Não é possível alterar o progresso deste item.

Reproduzir

115. O que são referências?

10m

Recursos

* Aula concluída. Não é possível alterar o progresso deste item.

Reproduzir

116. Praticando: Criando e Utilizando Referências

6m

Recursos

* Aula concluída. Não é possível alterar o progresso deste item.

Reproduzir

117. Consolidando os conceitos de Referências

6m

* Aula concluída. Não é possível alterar o progresso deste item.

Reproduzir

118. Passagem de argumentos para funções por valor

9m

Recursos

* Aula concluída. Não é possível alterar o progresso deste item.

Reproduzir

119. Passagem de argumentos para funções por referência utilizando referências

7m

Recursos

* Aula concluída. Não é possível alterar o progresso deste item.

Reproduzir

120. Passagem de argumentos para funções por referência utilizando ponteiros

9m

Recursos

* Aula concluída. Não é possível alterar o progresso deste item.

Iniciar

121. Explicando um pouco mais sobre passagem de argumentos por valor e por referência

6m

* Aula não concluída. Não é possível alterar o progresso deste item.

Iniciar

Teste 7: Teste seus conhecimentos sobre Referências

Seção 11: Vetores(Arrays) e Matrizes

0 / 21 | 2h 16m0 de 21 aulas concluídas2h 16m

Seção 12: Aritmética de Ponteiros e a relação com Vetores \*(ptr + 2) , ptr++, ptr--, \*ptr

0 / 6 | 45m0 de 6 aulas concluídas45m

Seção 13: Programação Orientada a Objetos em C++ : Classes, Objetos, Atributos e Métodos.

0 / 9 | 1h 3m0 de 9 aulas concluídas1h 3m

Seção 14: Entendendo as Regiões da RAM (Stack e Heap) para entender Orientação à Objetos

0 / 5 | 42m0 de 5 aulas concluídas42m

Seção 15: Orientação à Objetos: .h x .cpp - Ponteiro this, Construtores e Destrutores

0 / 7 | 1h 8m0 de 7 aulas concluídas1h 8m

Seção 16: Structs - Estruturas em C++

0 / 19 | 2h 2m0 de 19 aulas concluídas2h 2m

Seção 17: Conceitos Adicionais de Orientação a Objetos com C++

0 / 7 | 53m0 de 7 aulas concluídas53m

Visão geral

Perguntas e respostasPerguntas e respostas

Observações

Anúncios

Avaliações

Ferramentas de aprendizado

[**Voltar para Todas as perguntas**](https://xpi.udemy.com/course/cmaismaisbasico/learn/#questions)