Workshop C - Aula 01

Fundamentos da linguagem C
27/09/2025

Samuel Abrão

IDP
abrao.sam.2006@gmail.com
github.com/samuka7abr

Metodologia dos Aulões

- **Explicação teórica**
- **Exercícios modelo de programação competitiva** (Codeforces)
- **Ponto extra nas matérias:** EDA & APC
- **Monitores para auxílio**

A História do C

A História do C

- Criada em 1972 por Dennis Ritchie

A História do C

- Criada em 1972 por Dennis Ritchie
- Evolução da linguagem B (fraca e com muitos furos)

A História do C

- Criada em 1972 por Dennis Ritchie
- Evolução da linguagem B (fraca e com muitos furos)
- Refatoração do sistema Unix (1973) (Antes escrito em ASM)

Aplicações em C

Aplicações em C

- Linux (o maioral)

Aplicações em C

- Linux (o maioral)
- Compiladores

Aplicações em C

- Linux (o maioral)
- Compiladores
- MySQL & PostgreSQL

Aplicações em C

- Linux (o maioral)
- Compiladores
- MySQL & PostgreSQL
- Sistemas embarcados

Aplicações em C

- Linux (o maioral)
- Compiladores
- MySQL & PostgreSQL
- Sistemas embarcados
- Jogos (Doom, Quake)

Por que aprender C?

Por que aprender C?

C não foi feito para ser simples!

- Tipagem forte e rigorosa, o que incentiva boas práticas e disciplina no código.

Por que aprender C?

C não foi feito para ser simples!

- Tipagem forte e rigorosa, o que incentiva boas práticas e disciplina no código.
- Foco nos fundamentos, permitindo compreender de forma clara e direta os conceitos essenciais.

Por que aprender C?

C não foi feito para ser simples!

- Tipagem forte e rigorosa, o que incentiva boas práticas e disciplina no código.
- Foco nos fundamentos, permitindo compreender de forma clara e direta os conceitos essenciais.
- Pouca abstração, obrigando o estudante a entender os conceitos desde a base.

Por que aprender C?

C não foi feito para ser simples!

- Tipagem forte e rigorosa, o que incentiva boas práticas e disciplina no código.
- Foco nos fundamentos, permitindo compreender de forma clara e direta os conceitos essenciais.
- Pouca abstração, obrigando o estudante a entender os conceitos desde a base.
- Próxima do hardware, sendo a linguagem de alto nível que mais se aproxima da lógica de funcionamento do computador.

Compilação e Execução

Compilação e Execução

- Compilar → o compilador transforma o código em C em uma "tradução" que o computador entende.

Compilação e Execução

- Compilar → o compilador transforma o código em C em uma "tradução" que o computador entende.
- Linkar → junta essa tradução com outras partes prontas (bibliotecas) para formar o programa.

Compilação e Execução

- Compilar → o compilador transforma o código em C em uma "tradução" que o computador entende.
- Linkar → junta essa tradução com outras partes prontas (bibliotecas) para formar o programa.
- Executar → o computador roda o programa e mostra o resultado.

Tipos Primitivos

Tipos Primitivos em C

```c
#include <stdio.h>

int main() {
    int idade = 20;
    printf("Idade: %d\n", idade);
```

Tipos Primitivos em C

```c
#include <stdio.h>

int main() {
    int idade = 20;
    printf("Idade: %d\n", idade);

    float altura = 1.75f;
    printf("Altura: %.2f\n", altura);
```

Tipos Primitivos em C

```c
#include <stdio.h>

int main() {
    int idade = 20;
    printf("Idade: %d\n", idade);

    float altura = 1.75f;
    printf("Altura: %.2f\n", altura);

    double pi = 3.1415926535;
    printf("PI: %.10f\n", pi);
```

Tipos Primitivos em C

```c
#include <stdio.h>

int main() {
    int idade = 20;
    printf("Idade: %d\n", idade);

    float altura = 1.75f;
    printf("Altura: %.2f\n", altura);

    double pi = 3.1415926535;
    printf("PI: %.10f\n", pi);

    char letra = 'A';
    printf("Letra: %c\n", letra);

    return 0;
}
```

Tipos Primitivos em C

```c
#include <stdio.h>

int main() {
    long populacao = 8000000000;
    printf("População: %ld\n", populacao);
```

Tipos Primitivos em C

```c
#include <stdio.h>

int main() {
    long populacao = 8000000000;
    printf("População: %ld\n", populacao);

    unsigned int pontos = 100;
    printf("Pontos: %u\n", pontos);

    return 0;
}
```

Representação em bits

Representação em bits

- **char**: 8 bits (1 byte)

Representação em bits

- **char**: 8 bits (1 byte)
- **int**: 32 bits (4 bytes)

Representação em bits

- **char**: 8 bits (1 byte)
- **int**: 32 bits (4 bytes)
- **float**: 32 bits (4 bytes)

Representação em bits

- **char**: 8 bits (1 byte)
- **int**: 32 bits (4 bytes)
- **float**: 32 bits (4 bytes)
- **double**: 64 bits (8 bytes)

Representação em bits

- **long**: 64 bits (8 bytes)

Representação em bits

- **long**: 64 bits (8 bytes)
- **unsigned int**: 32 bits (4 bytes)

Representação em bits

- **long**: 64 bits (8 bytes)
- **unsigned int**: 32 bits (4 bytes)
- **short**: 16 bits (2 bytes)

Representação em bits

- **long**: 64 bits (8 bytes)
- **unsigned int**: 32 bits (4 bytes)
- **short**: 16 bits (2 bytes)
- **long long**: 64 bits (8 bytes)

Operadores aritméticos, lógicos e de atribuição

Operadores Aritméticos

```c
#include <stdio.h>

int main() {
    int a = 10, b = 3;

    int soma = a + b;          // Adição
    int sub = a - b;           // Subtração
    int mult = a * b;          // Multiplicação
    int div = a / b;           // Divisão
    int mod = a % b;           // Módulo (resto)

    return 0;
}
```

Operadores de Atribuição

```c
#include <stdio.h>

int main() {
    int a = 10, b = 3;

    a += 5;                    // At com adição
    b -= 2;                    // At com subtração
    a *= 3;                    // At com mult
    b /= 2;                    // At com divisão

    return 0;
}
```

Operadores Lógicos

```c
#include <stdio.h>

int main() {
    int x = 1, y = 0;

    int and_logico = x && y; // E lógico
    int or_logico = x || y;  // OU lógico
    int not_logico = !x;     // NÃO lógico

    return 0;
}
```

Operadores Relacionais

```c
#include <stdio.h>

int main() {
    int a = 10, b = 3;

    int maior = a > b;        // Maior que
    int menor = a < b;        // Menor que
    int igual = a == b;       // Igual a
    int diferente = a != b;   // Diferente de

    return 0;
}
```

Printf

printf - Básico

```c
#include <stdio.h>

int main() {
    printf("Olá, mundo!\n");
    return 0;
}
```

printf com Variáveis

```c
#include <stdio.h>

int main() {
    int idade = 20;
    float altura = 1.75;
    char nome[] = "João";

    printf("Nome: %s\n", nome);
    printf("Idade: %d\n", idade);
    printf("Altura: %.2f\n", altura);

    return 0;
}
```

printf com Formatação

```c
#include <stdio.h>

int main() {
    int idade = 20;
    char nome[] = "João";

    printf("Dados:\n\tNome: %s\n\tIdade: %d\n",
     nome, idade);

    return 0;
}
```

scanf

scanf - Entrada de Dados

```c
#include <stdio.h>
int main() {
    int idade;
    float altura;
    char nome[50];
    printf("Digite sua idade: ");
    scanf("%d", &idade);
    printf("Digite sua altura: ");
    scanf("%f", &altura);
    printf("Digite seu nome: ");
    scanf("%s", nome);
    printf("Nome: %s, Idade: %d, Altura: %.2f\n",
           nome, idade, altura);
    return 0;
}
```

Por que usar &?

- O **scanf** precisa alterar o conteúdo da variável

Por que usar &?

- O **scanf** precisa alterar o conteúdo da variável
- Em C, argumentos são passados por **cópia**

Por que usar &?

- O **scanf** precisa alterar o conteúdo da variável
- Em C, argumentos são passados por **cópia**
- **&idade** fornece o **endereço** da variável

Por que usar &?

- O **scanf** precisa alterar o conteúdo da variável
- Em C, argumentos são passados por **cópia**
- **&idade** fornece o **endereço** da variável
- Assim o scanf consegue gravar diretamente na memória

Por que char não precisa de &?

PQ O CHAR NÃO PRECISA DE "&"?

- **char nome[50]** já é um **ponteiro**

Por que char não precisa de &?

PQ O CHAR NÃO PRECISA DE "&"?

- **char nome[50]** já é um **ponteiro**
- O nome do array aponta para o primeiro elemento

Por que char não precisa de &?

PQ O CHAR NÃO PRECISA DE "&"?

- **char nome[50]** já é um **ponteiro**
- O nome do array aponta para o primeiro elemento
- **nome** = endereço do primeiro caractere

Por que char não precisa de &?

PQ O CHAR NÃO PRECISA DE "&"?

- **char nome[50]** já é um **ponteiro**
- O nome do array aponta para o primeiro elemento
- **nome** = endereço do primeiro caractere
- Por isso: **scanf("%s", nome)** funciona

math.h

Biblioteca math.h

Biblioteca padrão de C para operações matemáticas.

Deve ser incluída com:

```
#include <math.h>
```

Contém funções para:

- Potências e raízes
- Trigonometria
- Arredondamento
- Valor absoluto

Potência e Raiz

```
sqrt(x) → raiz quadrada de x.
pow(base, exp) → eleva base ao expoente exp.
```

Exemplo:

```
double r1 = sqrt(25);      // 5
double r2 = pow(2, 3);     // 8
```

Valor Absoluto e Arredondamento

```
fabs(x) → valor absoluto de um número real.
ceil(x) → arredonda para cima.
floor(x) → arredonda para baixo.
```

Exemplo:

```
double a = fabs(-7.3);     // 7.3
double b = ceil(4.2);      // 5
double c = floor(4.8);     // 4
```

Funções Trigonométricas

```
sin(x), cos(x), tan(x)
```

Argumentos em radianos.

Conversão de graus para radianos:

```
rad = graus * M_PI / 180;
```

Exemplo:

```
double r = sin(M_PI / 2); // 1
```

Dica de Compilação

A math.h precisa ser linkada manualmente no compilador GCC:

```
gcc programa.c -o programa -lm
```

Sem -lm, o programa pode não compilar corretamente.

Hora da prática!

Hora da prática!