
# Laboratório de Programação

Profº - Dr. Thales Levi Azevedo Valente

thales.l.a.valente@gmail.com

Governo Federal

# Sejam Bem-vindos !

**Os celulares devem ficar no silencioso ou desligados**

Pode ser utilizado apenas em caso de emergência

**Boa tarde/noite, por favor e com licença DEVEM ser usados**

Educação é essencial

# Na aula de anterior...

**Realizar uma dinâmica para conhecer um ao outro**

Discutir sonhos e desejos

A importância de ter um objetivo definido

**Discutir boas práticas de estudo**

Importância de um cronograma

Importância do foco

Importância de revisões periódicas

Alimentação e exercício

# Na aula de hoje...

**Avaliações**

Sala: Atividades(10%) presença (10%)

2 provas (40%) + 1 Trabalho(30%) + ?

# Objetivos de hoje

Apresentar conceitos introdutórios da linguagem C, ambientes de programação e rodar alguns códigos

Ao final da aula, os alunos conhecerão um pouco do “Ambiente C” e serem capazes de rodar alguns códigos☺

# Roteiro: Aula de Introdução

- Características da Linguagem
- Linguagem Compilada
- Linguagem Alto Nível
- Linguagem Portável
- Estrutura Básica
  - Sintaxe
  - Semântica
  - Sugestões de IDE
- C vs Assembly
- C vs Java
- C vs Python

# História

- Primeira atividade: Texto escrito para a próxima aula destancando os seguintes pontos
  - Origem e desenvolvimento da linguagem C
  - Influência da linguagem C no mundo da programação
  - Relação entre Dennis Ritchie e o desenvolvimento Linux

# Caracterísitcas

- Alto Nível
- Compilada
- Portável

# Características – Compilada

```c
#include <stdio.h> // Incluir a biblioteca de entrada e saída

int main() { // Função principal
    printf("Olá, Mundo!\n"); // Imprimir "Olá, Mundo!" na tela
    return 0; // Retornar 0 (sucesso)
}
```

- Código-Fonte: o processo começa com a escrita do código-fonte em um arquivo de texto com a extensão .c, usando um editor de texto ou uma IDE.

- Compilador:
  - Cerifica o código-fonte em busca de erros
  - Um compilador C (por exemplo, gcc) transforma o código-fonte em um arquivo objeto (.o ou .obj).
  - Este arquivo objeto contém código de máquina, mas ainda não está em um formato que pode ser executado independentemente

# Características – Compilada

```c
#include <stdio.h> // Incluir a biblioteca de entrada e saída

int main() { // Função principal
    printf("Olá, Mundo!\n"); // Imprimir "Olá, Mundo!" na tela
    return 0; // Retornar 0 (sucesso)
}
```

```
Código Fonte (.c)
    ↓ (Compilação)
Arquivo Objeto (.o ou .obj)
    ↓ (Linkagem)
Arquivo Executável
    ↓ (Execução)
Saída do Programa
```

- Linkagem:
  - Um linker combina o arquivo objeto com bibliotecas de código adicionais para criar um arquivo executável
  - O arquivo executável contém código de máquina que pode ser executado diretamente pelo sistema operacional
- Execução:
  - Finalmente, o arquivo executável é executado para realizar a tarefa para a qual o programa foi escrito

# Características – Alto Nível

- Facilidade de aprendizado: são mais abstratas e estão mais distantes do hardware. São projetadas para serem mais fáceis de aprender e usar.

- Portabilidade: podem ser executados em diferentes sistemas e arquiteturas com poucas ou nenhuma modificação, desde que exista um compilador ou interpretador para cada plataforma

- Produtivas: permitem aos desenvolvedores escrever código de forma mais eficiente, sem se preocupar com detalhes específicos do hardware

# Características – Alto Nível

**Quem é alto e baixo nível?**

hello.S

```assembly
; Exemplo de um Hello World em Assembly
; ld -m elf_i386 -s -o hello hello.o
section .text align=0

global _start

mensagem db 'Hello world', 0x0a

len equ $ - mensagem

_start:
    mov eax, 4 ;SYS_write
    mov ebx, 1 ;Número do file descriptor (1=stdout)
    mov ecx, mensagem ;Ponteiro para a string.
    mov edx, len ; tamanho da mensagem
    int 0x80

    mov eax, 1
    int 0x8
```

Hello world em linguagem Assembly

```c
#include <stdio.h> // Incluir a biblioteca de entrada e saída

int main() { // Função principal
    printf("Olá, Mundo!\n"); // Imprimir "Olá, Mundo!" na tela
    return 0; // Retornar 0 (sucesso)
}
```

Hello world em linguagem C

# Características – Portabilidade

- Definição: refere-se à capacidade de um programa ser compilado e executado em diferentes plataformas (sistemas operacionais e arquiteturas de hardware) sem a necessidade de modificação no código-fonte.

- Importância: amplamente utilizada no desenvolvimento de sistemas operacionais, drivers de dispositivos e sistemas embarcados.

# Características – Portabilidade

- Independência de Plataforma de aprendizado: pode ser compilado em quase qualquer sistema operacional moderno e arquitetura de hardware, incluindo Windows, Linux, macOS, ARM, e outros.

- Padronização: programas escritos em C padrão devem ser compilados e executados da mesma maneira em qualquer sistema que siga esses padrões.

- Compiladores Disponíveis: compiladores para a ampla variedade de sistemas operacionais e hardware, facilitando a compilação de código C em diferentes plataformas

# Curiosidade – Comparação: C vs Python

- C:
  - compilada, ou seja, código traduzido diretamente no código de máquina
  - Desempenho mais rápido em tarefas intensivas
  - Gerenciamento de memória mais manual
  - Procedural
  - Tipos devem ser declarados

- Python:
  - Sendo interpretada, executa mais lentamente que código compilado.
  - Mais fácil de aprender / usar
  - Gerenciamento de memória mais automático
  - Orientada a objetos
  - Tipagem automática

# Curiosidade – Comparação: C vs Java

- C:
  - compilada, ou seja, código traduzido diretamente no código de máquina
  - Desempenho mais rápido em tarefas intensivas
  - Gerenciamento de memória mais manual
  - Procedural
  - Tipos devem ser declarados

- Java:
  - Compilada, mas precisa de uma máquina virtual para o ambiente alvo.
  - Mais lento (pelo passo intermediário)
  - Gerenciamento de memória mais automático
  - Orientada a objetos
  - Tipos devem ser declarados

# Curiosidade – Comparação: C vs Assembly

- C:
  - compilada, ou seja, código traduzido diretamente no código de máquina
  - Desempenho mais rápido em tarefas intensivas
  - Gerenciamento de memória mais manual
  - Procedural
  - Tipos devem ser declarados

- Assembly:
  - É a própria linguagem de máquina.
  - Nada é mais rápido
  - Gerenciamento de memória “hard”
  - Total controle sobre o hardware
  - Complexa e trabalhosa para escrever e depurar

# Estrutura Básica

- Bibliotecas
- Função MAIN
- Comentários
- Linha de código e ponto-e-vírgula
- Bloco de código

# Bibliotecas – O que são?

- Bibliotecas em C são conjuntos de funções, macros e definições de tipos que fornecem funcionalidades úteis, permitindo aos programadores realizar tarefas comuns sem ter que escrever seu próprio código do zero

- Bibliotecas em C são conjuntos de funções, macros e definições de tipos que fornecem funcionalidades úteis, permitindo aos programadores realizar tarefas comuns sem ter que escrever seu próprio código do zero

# Bibliotecas – Uso

- Para usar uma biblioteca em um programa C, os programadores geralmente usam uma diretiva #include no início do código-fonte.

- Isso instrui o compilador a incluir as definições e funções da biblioteca no programa durante a compilação.

# Bibliotecas – Importância

- As bibliotecas economizam tempo e esforço, fornecendo implementações prontas para uma ampla variedade de tarefas.

- As bibliotecas economizam tempo e esforço, fornecendo implementações prontas para uma ampla variedade de tarefas.

# Bibliotecas – exemplos

- **<stdio.h>**: fornece funcionalidades de entrada e saída.
  - Funções comuns: printf(), scanf(), fgets(), fputs()
- **<stdlib.h>:** funções de alocação de memória, controle de processo, conversões de dados
  - Funções comuns: malloc(), free(), exit(), atoi()
- **<string.h>:** funções para manipular strings
  - Funções comuns: strcpy(), strcat(), strlen(), strcmp(), etc.

```c
#include <stdio.h>

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

```c
#include <stdlib.h>

int main() {
    int *ptr = (int*)malloc(sizeof(int) * 5); //
    if (ptr != NULL) {
        // usa a memória alocada
        free(ptr); // libera a memória alocada
    }
    return 0;
}
```

```c
#include <string.h>

int main() {
    char str1[20] = "Olá, ";
    char str2[] = "Mundo!";
    strcat(str1, str2); // concatena str2 em str1
    return 0;
}
```

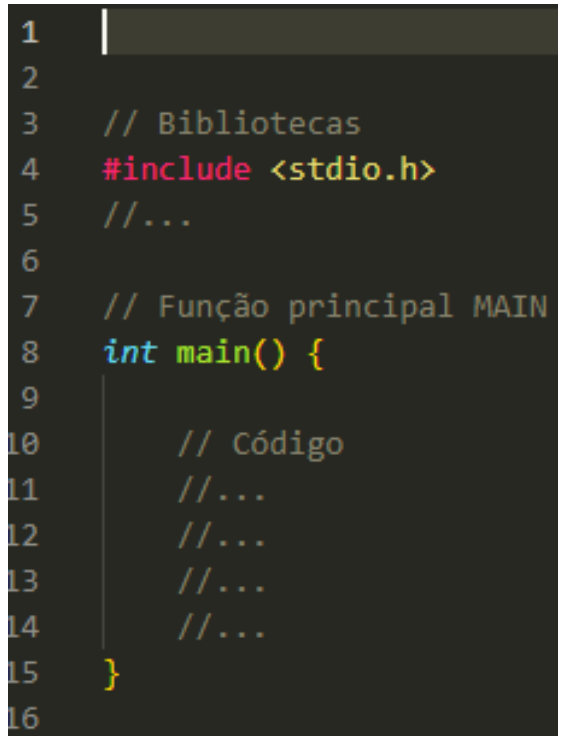
# – Importância

- As bibliotecas economizam tempo e esforço, fornecendo implementações prontas para uma ampla variedade de tarefas.

- As bibliotecas economizam tempo e esforço, fornecendo implementações prontas para uma ampla variedade de tarefas.

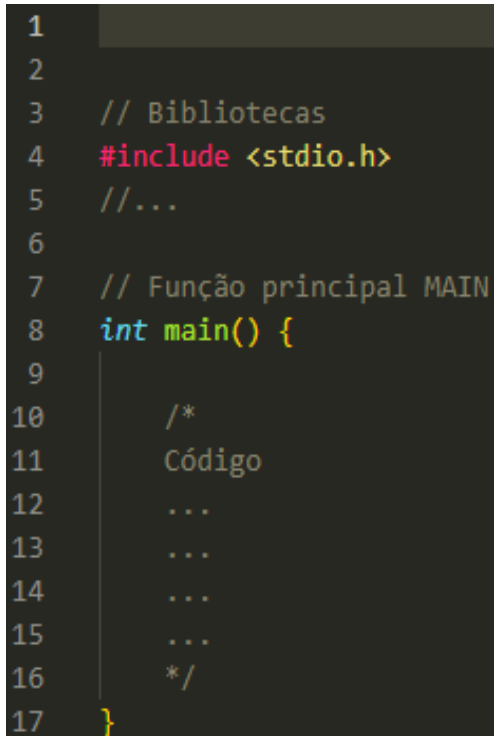
# Função MAIN

- Ponto de partida da execução de um código ou programa.
- Função inicial
- A partir dela todo restante do código é executado
- A Função MAIN pode conter outras funções
- Função “Mãe”



# Comentários

- Usados para adicionar descrições e explicações no código.
- Não são processados pelo compilador.
- Comentários de uma linha são precedidos por // e comentários de várias linhas são delimitados por /* e */.



# Bloco de código

- Blocos de código são delimitados por chaves {}.
- São utilizados para agrupar várias instruções em uma única unidade lógica
- São essenciais em funções*, loops* e condições*
- Identifique os blocos de códigos nos exemplos

```c
if (numero > 10) {
    printf("O número é maior que 10.\n");
    numero = 10;
}
```

```c

// Bibliotecas
#include <stdio.h>
//...

// Função principal MAIN
int main() {

    /*
    Código
    ...
    ...
    ...
    ...
    */
}
```

# Linha de código e ponto-e-vírgula

- Qualquer linha “executável” pelo compilador
- Importação de bibliotecas
- Retorno de funções
- Atribuição de valores a variáveis
- Chamada de funções

```c
#include <string.h>

int main() {
    char str1[20] = "Olá, ";
    char str2[] = "Mundo!";
    strcat(str1, str2); // concatena str2 em str1
    return 0;
}
```

# Sintaxe

- Conjunto de regras que define a combinação correta de símbolos (palavras-chave, operadores, identificadores, etc.) que formam um programa válido ou expressão na linguagem
- Regras Formais:
  - Especifica como os símbolos da linguagem podem ser combinados.
- Erros de Sintaxe:
  - Violações das regras sintáticas são chamadas de erros de sintaxe e são captadas em tempo de compilação.
- Exemplos:
  - Posição de chaves ({}), uso correto de ponto e vírgula (;), formatação de declarações e expressões, etc.

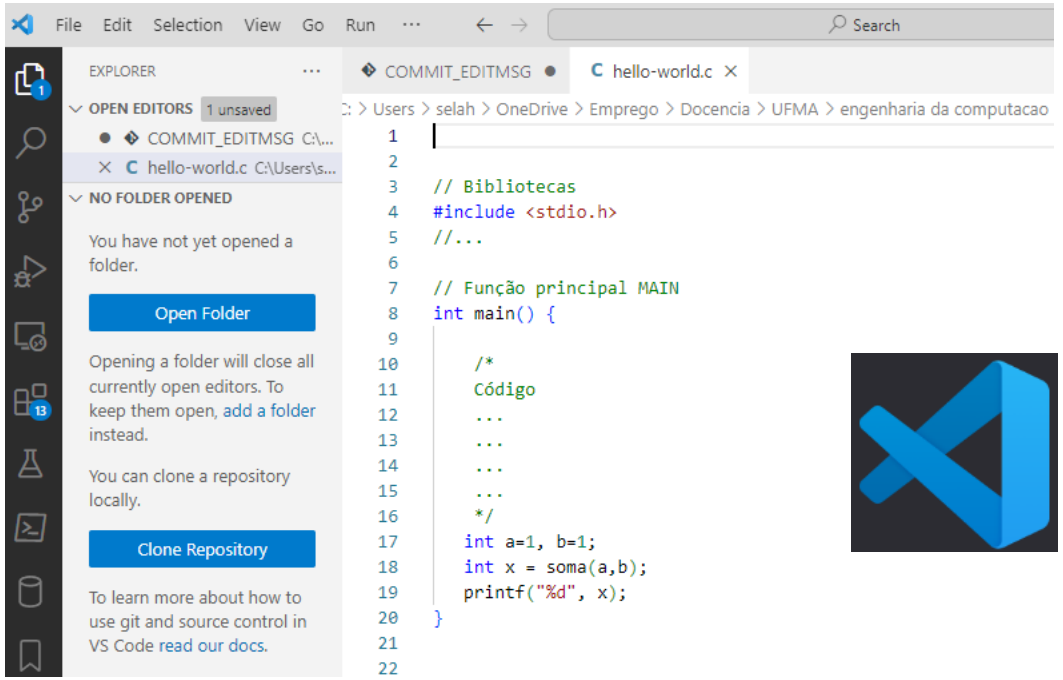
# Semântica

- Significado do programa ou expressão. Ela se preocupa com o que o programa faz durante a execução
- Comportamento em Tempo de Execução
  - Define o comportamento do programa em tempo de execução.
- Erros Semânticos
  - Violações das regras semânticas são chamadas de erros semânticos e muitas vezes só são detectadas em tempo de execução.
- Exemplos
  - Uso correto de tipos de dados, escopo de variáveis, passagem de argumentos para funções, etc.

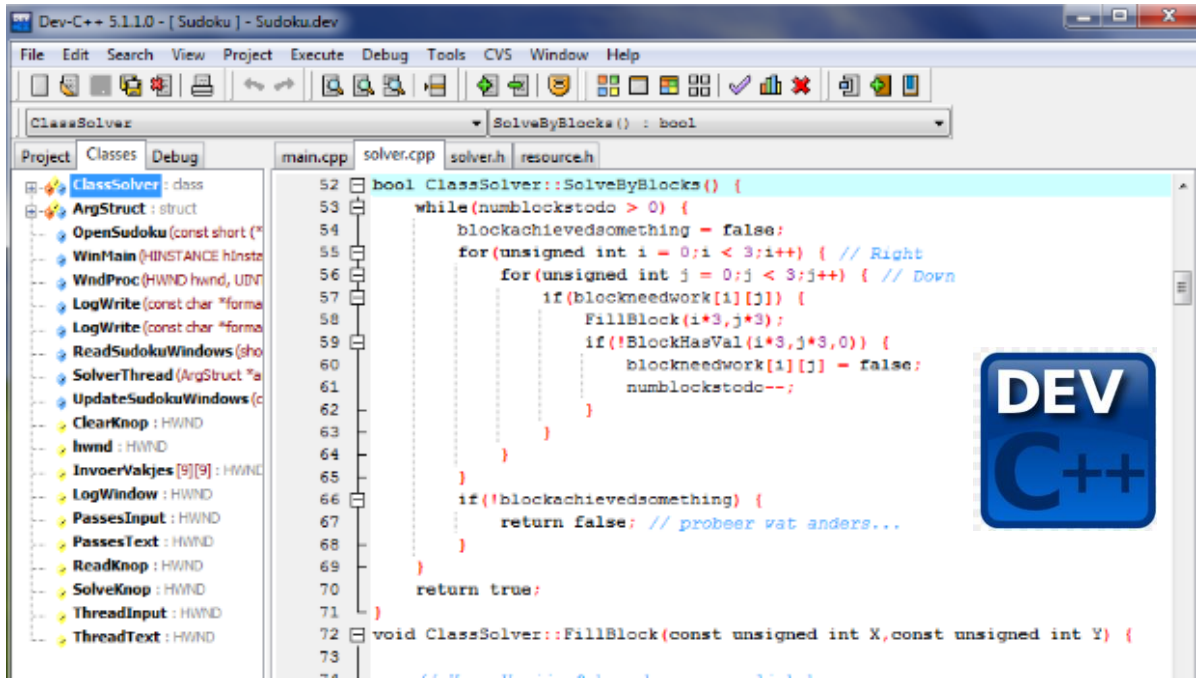
# Sintaxe vs Semântica

- Foco
  - Sintaxe: Foca na forma e estrutura do código.
  - Semântica: Foca no significado e comportamento do código.
- Erros
  - Sintaxe: Erros de sintaxe são detectados pelo compilador.
  - Semântica: Erros semânticos podem não ser detectados pelo compilador e podem causar comportamento inesperado em tempo de execução.
- Aspecto
  - Sintaxe: Trata das regras para escrever código correto.
  - Semântica: Trata do significado do código escrito.

# Sugestão de IDEs e Links para instalação

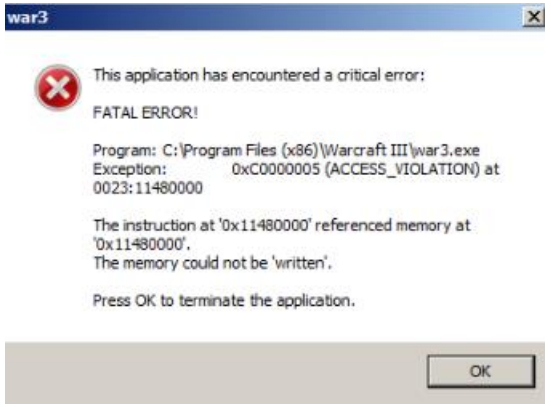



Get Started with C++ and MinGW-w64 in Visual Studio Code




How to Download and Install Dev C++ on Windows? - GeeksforGeeks

# Erro de semantica




Acessos acidentais à memória

Erro semântico "clássico" em *C*

```c
if (x = 0) // supondo que x chegue aqui com valor 0
  printf("O valor e' nulo\n");
```

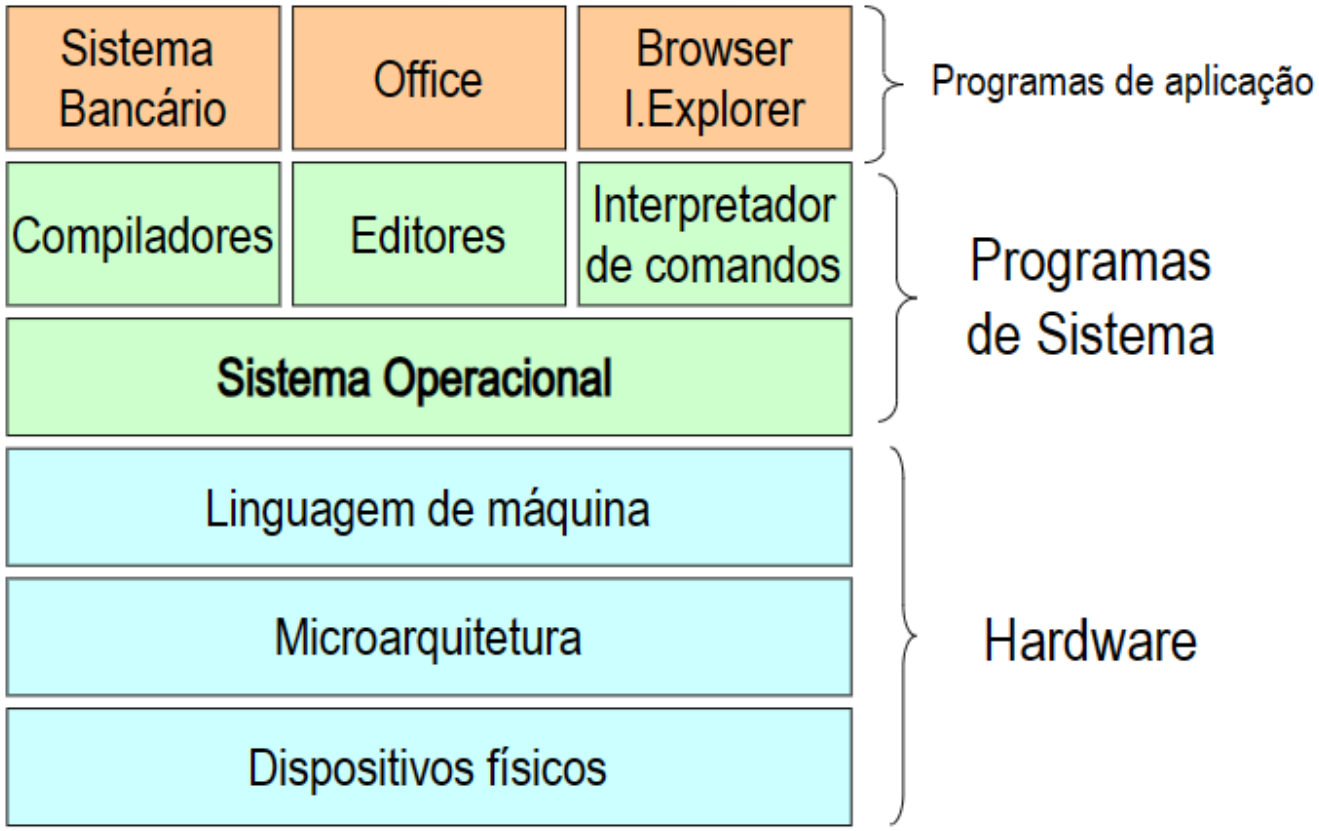
Comportamento inesperado

*C*

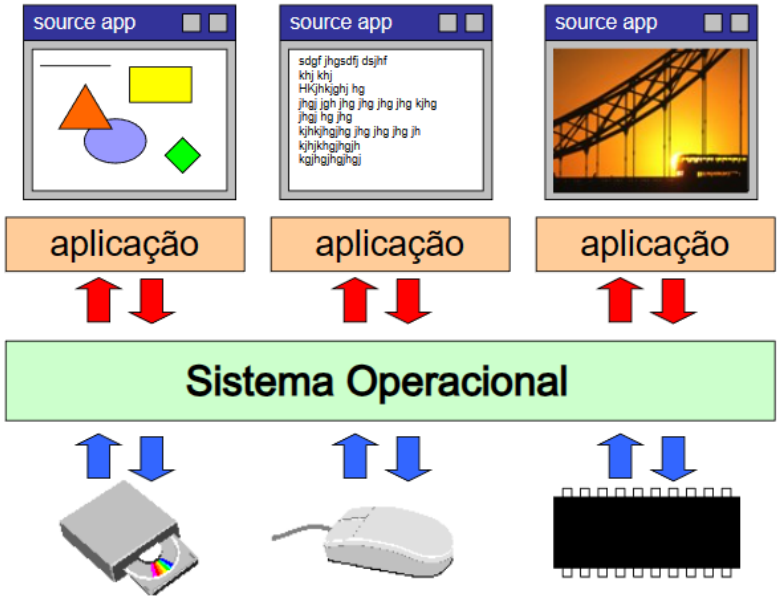
```c
#include <stdio.h>
int main (void) {
  int i=0;
  while (i < 10) {
    printf("hello world\n");
    }
  }
```

Execucao infinita

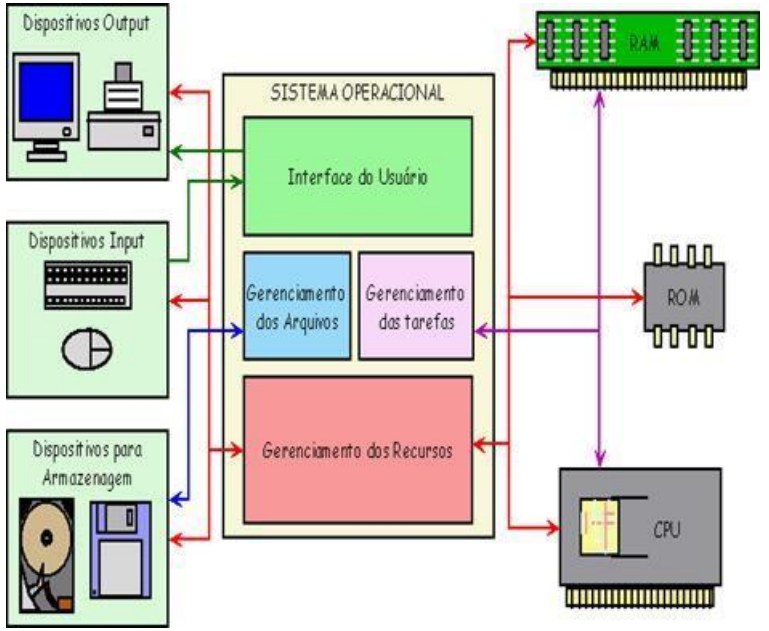
# Onde esta meu programa?



# Onde esta meu programa?



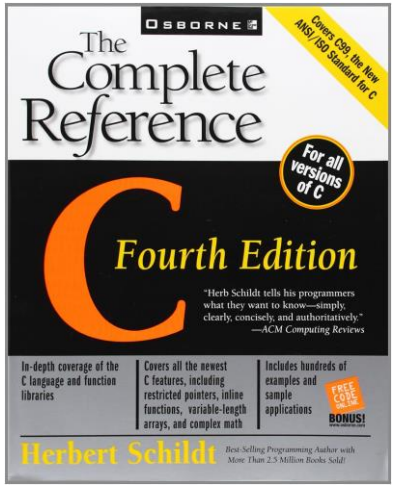

Visão em nível de aplicação




Visão em nível Geral

# Bibliografia

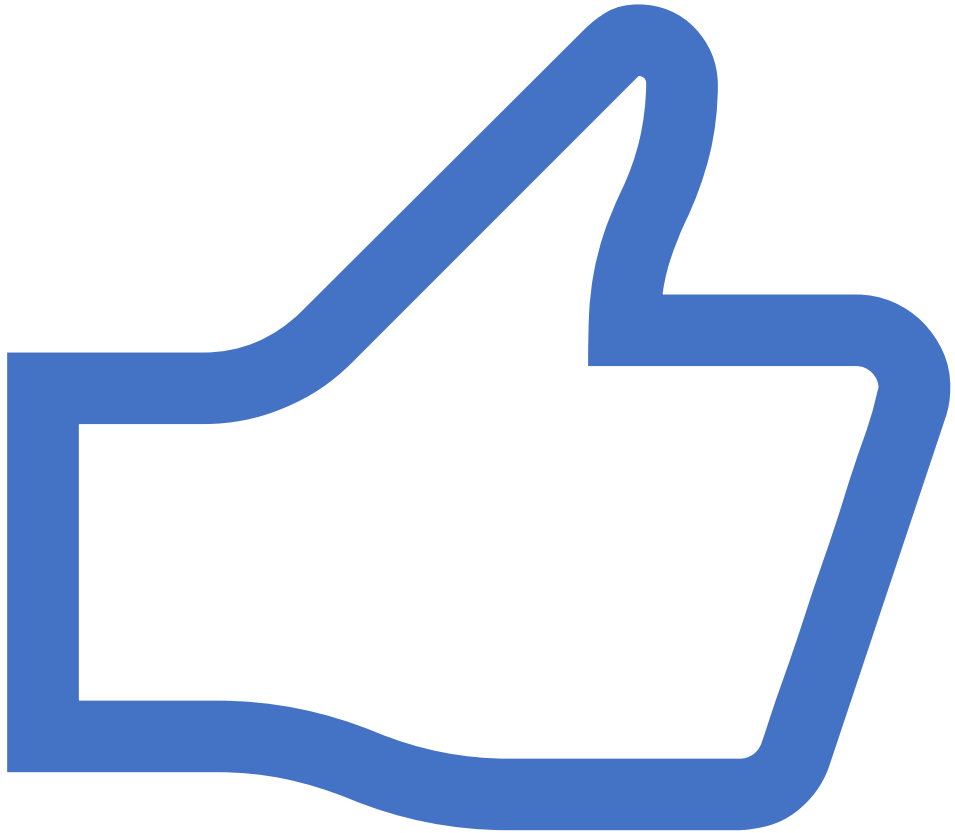




# Exemplos

# Dúvidas?

Obrigado !



Apresentador

# Thales Levi Azevedo Valente

E-mail:
thales.l.a.valente@gmail.com