CURSO SUPERIOR DE ADS

Apresentação da Disciplina e Paradigmas de Programação

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS

Apresentação da Disciplina

1. **Objetivo Geral:**

   - Construir uma base teórico-prática que possibilite ao discente desenvolver competências em programação estruturada, utilizando conceitos básicos, estruturas de controle, modularização e manipulação de dados, capacitando-o a implementar e solucionar problemas computacionais de maneira eficiente e organizada.

2. **Específicos:**

   - Compreender o paradigma de programação estruturada.
   - Realizar a implementação de pequenos projetos de software visando exercitar comandos e declarações.
   - Aplicar estruturas de controle e variáveis com clareza e eficiência.
   - Entender o mecanismo de modularização de sistemas.

Lembrando nossa ementa

Ementa: Conceitos básicos de programação estruturada em linguagem C. Ambientes de desenvolvimento (IDEs e compiladores). Comandos de entrada e saída. Tipos de dados, variáveis, constantes e casting. Estruturas de decisão e repetição. Estruturas de dados compostas (vetores e matrizes). Modularização com funções e procedimentos.

Bibliografia

Básica:

- BORIN, V. P. Estrutura de Dados.1ª Ed . Curitiba: Contentus, 2020. [acervo digital]
- FERREIRA, Ronaldo Domingues. Linguagem de Programação. Curitiba: Contentus, 2020. [acervo digital]
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: A construção de algoritmos e estruturas de dados com aplicações em Python. 4ª Ed. São Paulo: Pearson Education do Brasil, 2022. [acervo digital]

Complementar:

- ARAÚJO, S. Lógica de Programação e algoritmos. 1ª Ed. Curitiba: Contentus, 2020. [acervo digital]
- ASCENCIO, AFG, ARAUJO, GS. Estrutura de Dados: Algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo: Pearson Education do Brasil, 2010. [acervo digital]
- ADAMI, A. G.; MARTINOTTO, A. L.; KOLIVER, C.; CASSOL, L. A.; DORNELES, R. V; GAVA, V. Introdução à construção de algoritmos. 1ª Ed. Caxias do Sul, 2009. [acervo digital]
- MENEZES, A. M. Os paradigmas de aprendizagem de algoritmo computacional. 1ª Ed. São Paulo: Blucher Open Acess, 2015. [acervo digital]
- PUGA, Sandra Gavioli; RISSETTI, Gerson. Lógica de programação e estruturas de dados, com aplicações em Java. 3. ed. São Paulo, SP: Pearson, 2016.

Avaliações

1ª Unidade: Prova - 100% **Data:** 30/03

2ª Unidade: Prova (60%) + Projetos (40%) **Data:** 11/05

3ª Unidade: Prova (80%) + Pratica Integradora (20%) **Data:** 08/06

Media Final: ((1U + 2U + 3U) / 3) => 7 -> **Aprovado(a)**

Prova Final (Data) -> 22/06 ☹

Ambientes de Desenvolvimento Integrado (IDEs)

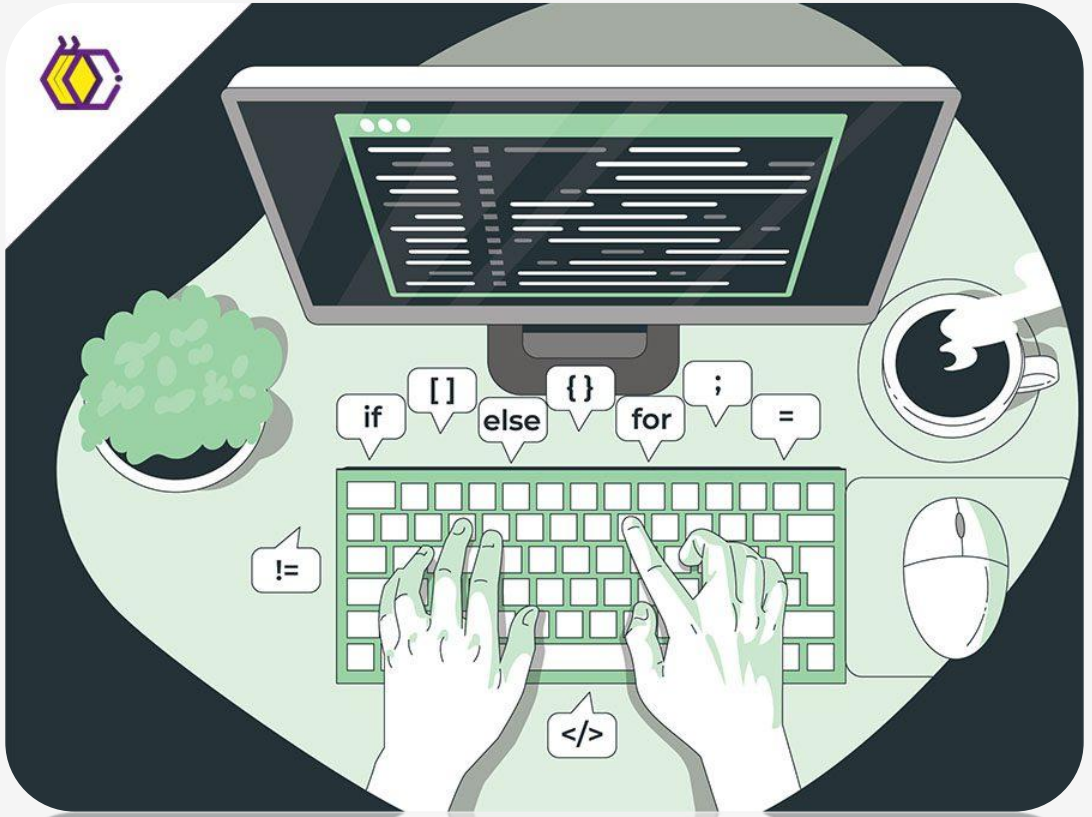
- **Definição:**
  - Ambientes de Desenvolvimento Integrado (IDEs) são plataformas que reúnem editor de código, compilador, depurador e outras funcionalidades em uma única interface.
- **Eficiência e Produtividade:**
  - Aumentam a produtividade
  - Oferecem autocompletar
  - Realce de sintaxe
  - Integração com sistemas de controle de versão

IDEs que serão utilizadas

- **Replit** → https://replit.com/
- **Dev-C++** → https://www.bloodshed.net/
- **VSCode** → https://code.visualstudio.com/

Paradigmas de Programação

- **Definição:** Linguagem de Programação (LP) é a notação formal para descrição de algoritmos em um computador
- **Importância:** entender a organização das linguagens e seus conceitos abstratos

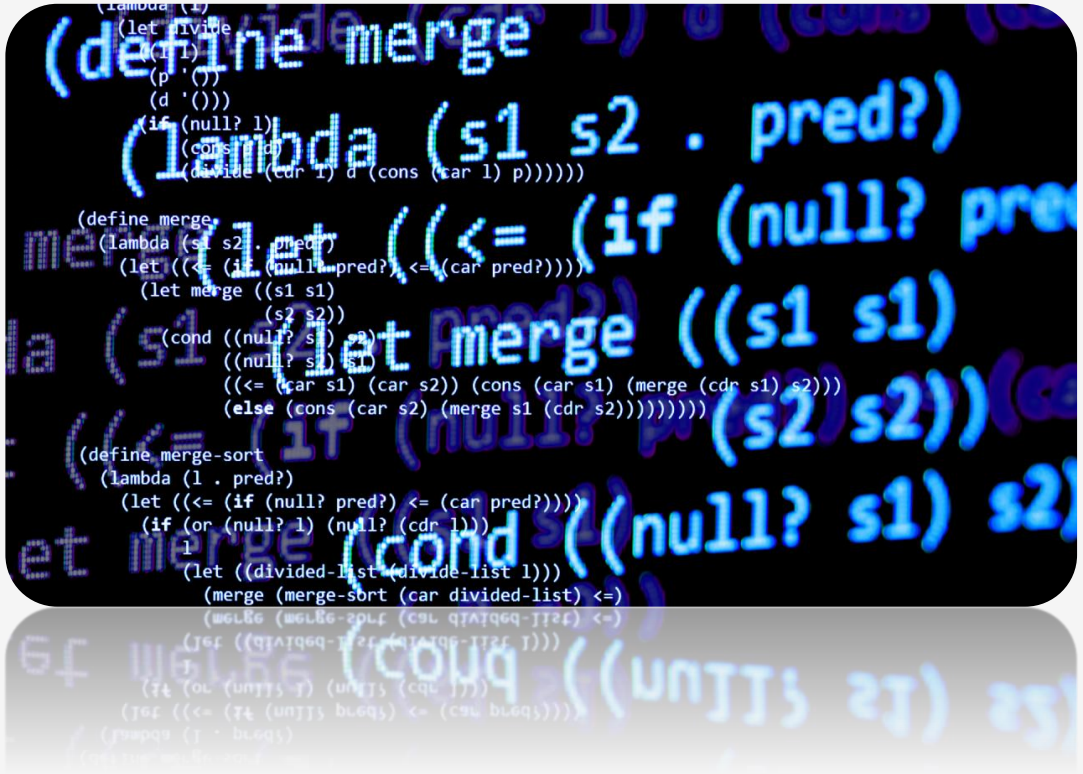


Programação Imperativa

- **Características:** instrui explicitamente como executar tarefas
- **Exemplo:** “Busque o usuário 15 no banco de dados. Valide essas informações”
- **Abordagem:** execução passo a passo
- **Problema:** pode ser mais propensa a erros se a lógica não estiver clara

Paradigmas principais

- **Programação Estruturada (Procedural):**
  - Usa estruturas sequenciais, condicionais e de repetição
  - Proporciona clareza, modularidade e manutenção
- **Programação Orientada a Objetos (POO):**
  - Classes, objetos, herança, polimorfismo
  - Reutilização de código e modelagem próxima ao mundo real
  - Exemplos: Smalltalk, Eiffel, Java



Outros Paradigmas

- **Programação Orientada a Restrições** → relações e restrições entre variáveis
- **Programação Orientada a Aspectos** → separa preocupações transversais (ex.: logging)
- **Programação Imperativa** → instruções que modificam o estado do programa

Exercício (Paradigmas)

Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa. Acerca do paradigma de programação estruturada, analise as asserções a seguir:

I – É um estilo de programação convencional (descrita no aprendizado em algoritmos). Os programas são decompostos em "passos" de processamento e são utilizadas rotinas para modularização de passos específicos;

II – É um paradigma de programação que se refere ao uso de restrições na construção de relações entre variáveis. Consiste em especificar, para uma solução, que critérios (restrições) esta tem de cumprir.

III – Apresenta um modelo de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

a) I apenas b) II apenas c) III apenas d) I e III e) II e III

Exercício (Paradigmas)

Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa. Acerca do paradigma de programação estruturada, analise as asserções a seguir:

I – É um estilo de programação convencional (descrita no aprendizado em algoritmos). Os programas são decompostos em "passos" de processamento e são utilizadas rotinas para modularização de passos específicos;

II – É um paradigma de programação que se refere ao uso de restrições na construção de relações entre variáveis. Consiste em especificar, para uma solução, que critérios (restrições) esta tem de cumprir.

III – Apresenta um modelo de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

a) I apenas b) II apenas c) III apenas d) I e III e) II e III

PROGRAMAÇÃO ESTRUTURADA

- **NÓS VAMOS TRABALHAR COM LINGUAGEM C**

História do C

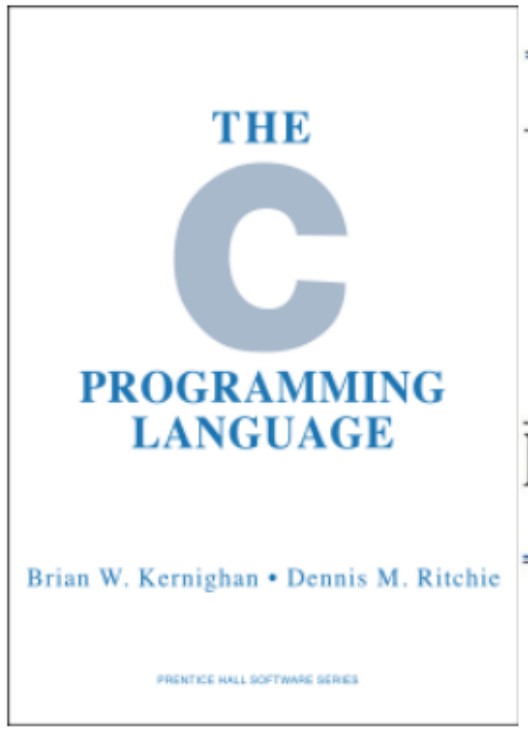
- A origem da linguagem C remonta ao início da década de 1970, quando um programador chamado **Dennis Ritchie**, que trabalhava nos Laboratórios Bell da AT&T (American Telephone and Telegraph), estava envolvido no desenvolvimento do sistema operacional Unix.
- O Unix, desenvolvido originalmente em assembly, estava se expandindo rapidamente, mas a programação em assembly era complexa e dependente do hardware específico.
- Por essa razão, Dennis Ritchie e **Ken Thompson** (co-criador do Unix) viram a necessidade de uma linguagem de programação mais portável e poderosa para continuar o desenvolvimento do sistema operacional.

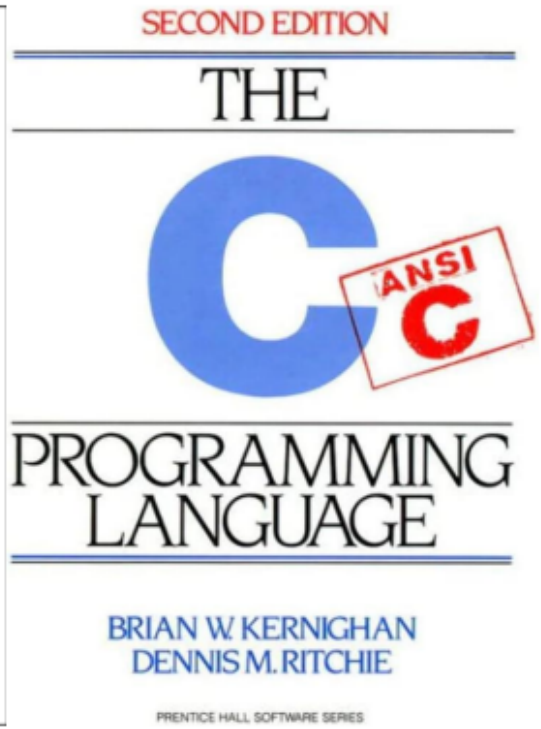
História do C

- Com base na linguagem de programação B (desenvolvida por Ken Thompson), Dennis Ritchie começou a trabalhar em uma nova linguagem que combinava recursos de portabilidade com uma abordagem de programação de mais alto nível.
- Essa linguagem foi originalmente chamada de "New B" ou "NB". Mais tarde, seu nome foi alterado para "C" devido a uma linguagem de programação anterior chamada BCPL (Basic Combined Programming Language).

Popularidade da Linguagem C

- C é derivada de duas linguagens do final da década de 60, o ALGOL68 e linguagem B (derivada do BCPL)
- Popularidade da linguagem C → cinco anos após lançamento, com publicação do livro ““The C programming Language”
- Padrão ANSI C estabelecido na década de 80 → modernização da linguagem (novas *libs, compatibilidade com C++ etc.)*





Algumas Curiosidades da Linguagem C

- Como **C é considerada a base de outras linguagens de programação**, se você puder aprender os conceitos usados nessa linguagem, será mais fácil entender outras linguagens mais tarde.
- C é uma ótima linguagem para programadores iniciantes. Não apenas porque a sintaxe é simples, mas porque C influenciou a maioria das linguagens mais utilizadas hoje em dia.
- Depois de aprender C, você descobre que ela tem muitos pontos em comum com Java, Javascript, Shell e PHP, por exemplo.

Princípios chave da linguagem C

Portabilidade: A intenção era que os programas escritos em C pudessem ser facilmente portados entre diferentes sistemas de computação sem a necessidade de reescrevê-los do zero.

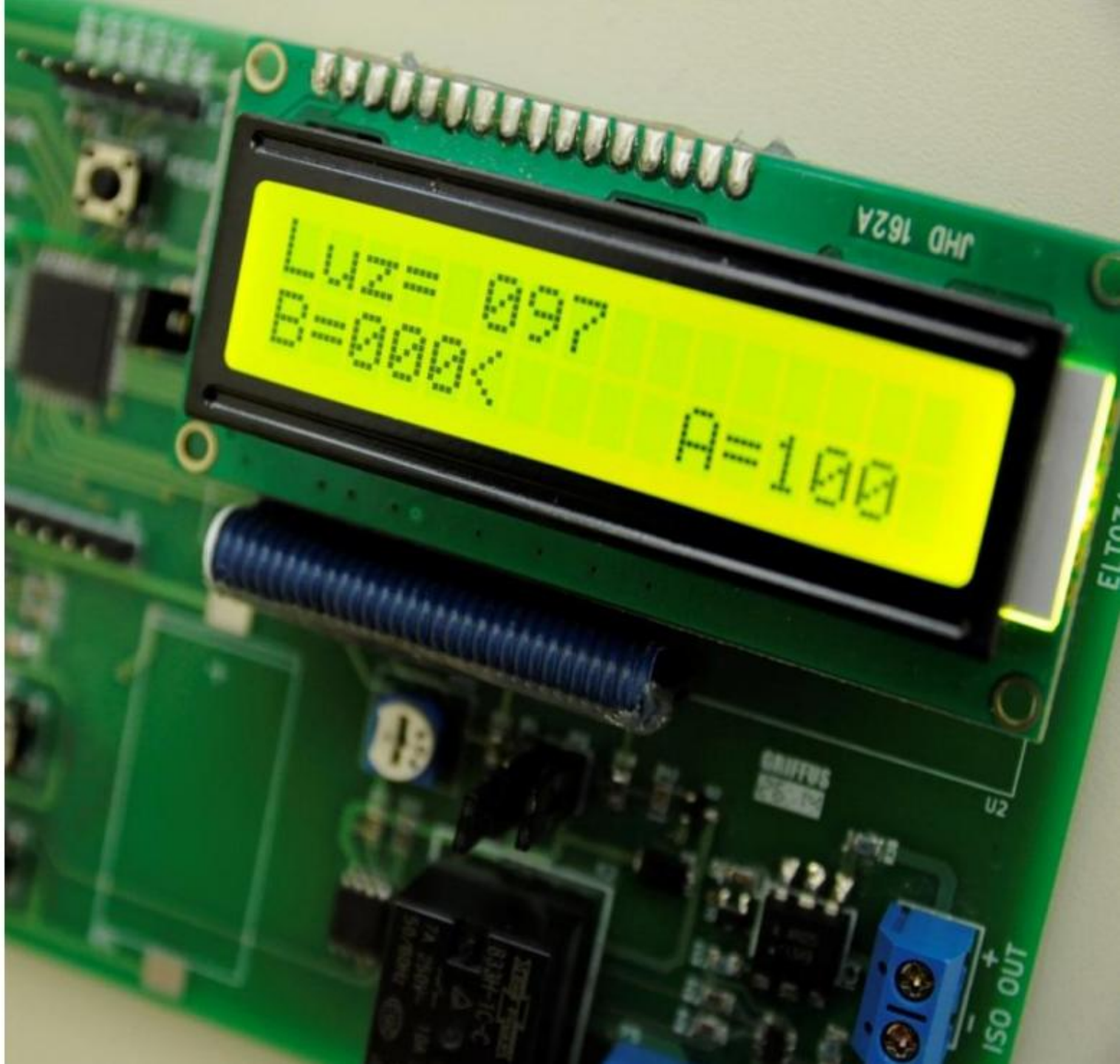
Eficiência: C foi projetada para permitir o acesso direto à memória e ao hardware, tornando-a uma linguagem de alto desempenho.

Baixo nível e alto nível: A linguagem oferece recursos de alto nível, como estruturas de controle e funções, mas também permite que os programadores escrevam código de baixo nível quando necessário, por meio de ponteiros, por exemplo.

Onde a linguagem C pode ser aplicada?

Sistemas embutidos/embarcados

- Digamos que todas as manhãs você use sua cafeteira para fazer o café. Logo, todos os dias você está utilizando um objeto que foi programado em C. Assim como a TV ou o sistema operacional do painel do carro que acionam o botão das travas de porta e vidro para crianças. Esses também são os chamados sistemas embutidos, programados em C. Quando você abre a porta da garagem com o controle remoto, também está usando um sistema embutido que provavelmente está programado em C.



Onde a linguagem C pode ser aplicada?

Os filmes 3D são criados com aplicativos ou softwares que geralmente são escritos em C e C++. Os softwares precisam ser muito eficientes e rápidos, pois lidam com uma enorme quantidade de dados e fazem muitos cálculos por segundo.

Quanto mais eficientes forem, menos tempo levará para os artistas e animadores gerarem as cenas do filme, e mais dinheiro a empresa economiza. Um exemplo é o jogo Quake II, programado em C.

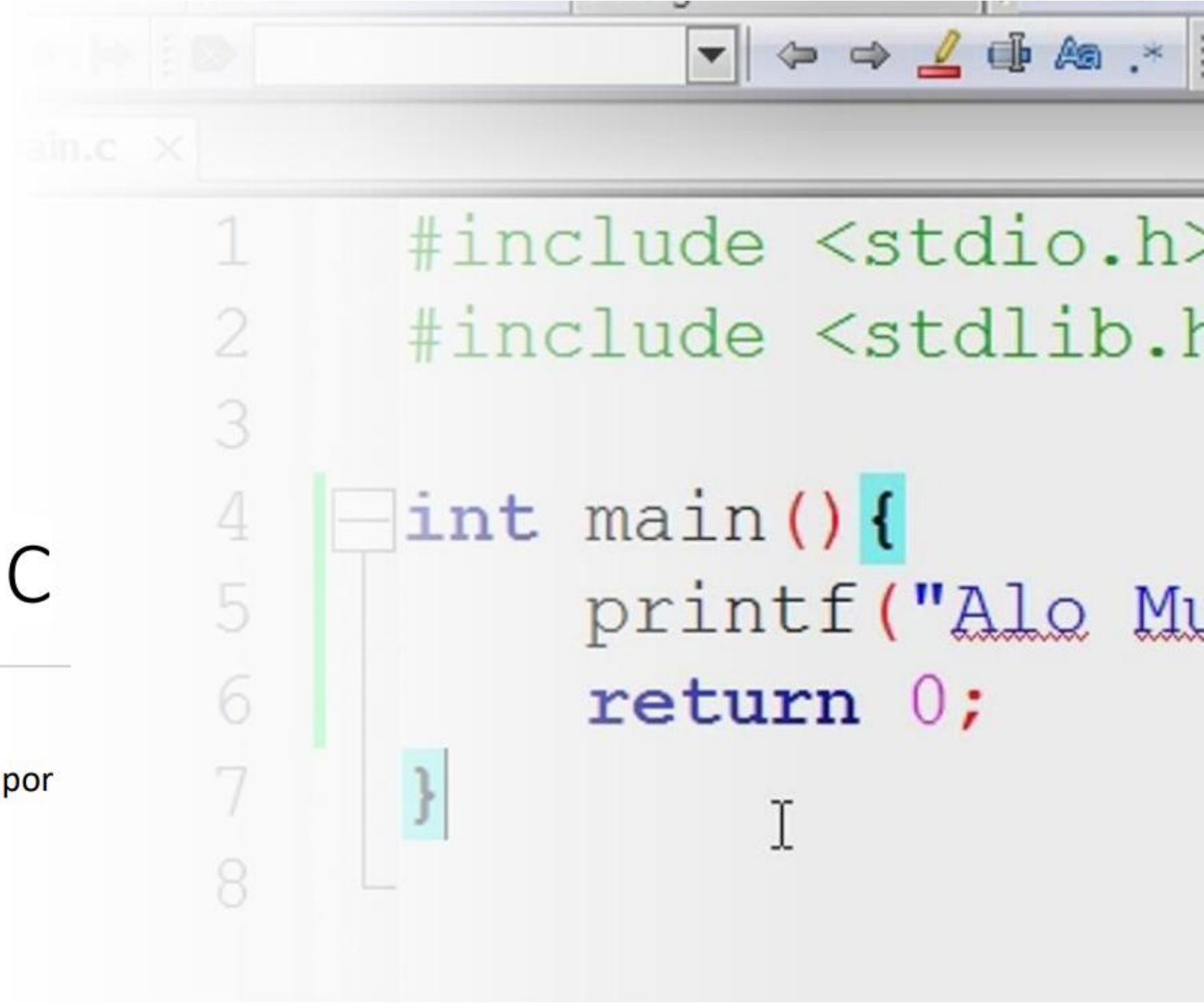
Onde a linguagem C pode ser aplicada?

Base de dados

- As bases de dados mais populares do mundo, incluindo Oracle Database, MySQL e MS SQL Server são programadas em C.

Estrutura de um programa em C

Um programa em C possui uma estrutura básica que é composta por diversos elementos essenciais.



1. Bibliotecas

- No início de um programa em C, são incluídas bibliotecas que fornecem funções e recursos pré-definidos para facilitar o desenvolvimento do código.

- Duas bibliotecas fundamentais são:

#include <stdio.h> // Biblioteca para funções de entrada/saída (I/O).

#include <stdlib.h> // Biblioteca para funções de utilidade geral.

stdio.h e stdlib.h

- A biblioteca **stdio.h** é amplamente utilizada para funções de entrada/saída, como printf() e scanf(), que permitem exibir informações na tela e capturar dados digitados pelo usuário.
- A biblioteca **stdlib.h** é frequentemente usada para funções como malloc() e free(), que lidam com alocação dinâmica de memória.

main.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello World\n");
    return 0;
}
```

2. Função main()

- Todo programa C deve conter uma função main(), que é o ponto de partida da execução do programa.
- A estrutura básica da função main() é a seguinte:

```
int main() {
    // Código do programa aqui

    return 0; // Indica que o programa foi executado com sucesso.
}
```

main()

- A função main() é obrigatória em um programa C e deve retornar um valor inteiro (int) para indicar o status de saída do programa.
- Geralmente, o valor 0 é retornado para indicar que o programa foi executado sem erros, e qualquer outro valor inteiro pode ser usado para indicar uma situação de erro específica.

main.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello World\n");
    return 0;
}
```

3. Comentários

- Comentários são trechos de texto que não são interpretados pelo compilador e servem para explicar o código, fazer anotações ou desabilitar temporariamente partes do programa.
- Em C, existem dois tipos de comentários:

```
// Comentário de linha única

/*
   Comentário
   de múltiplas
   linhas
*/
```

4. Estrutura de Bloco

- Em C, as instruções são agrupadas em blocos delimitados por chaves { }.
- A estrutura de bloco é usada para agrupar um conjunto de instruções e definir o escopo das variáveis.

main.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello World\n");
    return 0;
}
```

```c
int main() {
    // Início do bloco da função main

    // Código do programa aqui

    return 0; // Final do bloco da função main
}
```

Variáveis e Tipos de Dados

Variáveis

Uma variável em C é um espaço de memória com um nome associado, usado para armazenar valores temporários ou permanentes durante a execução do programa.

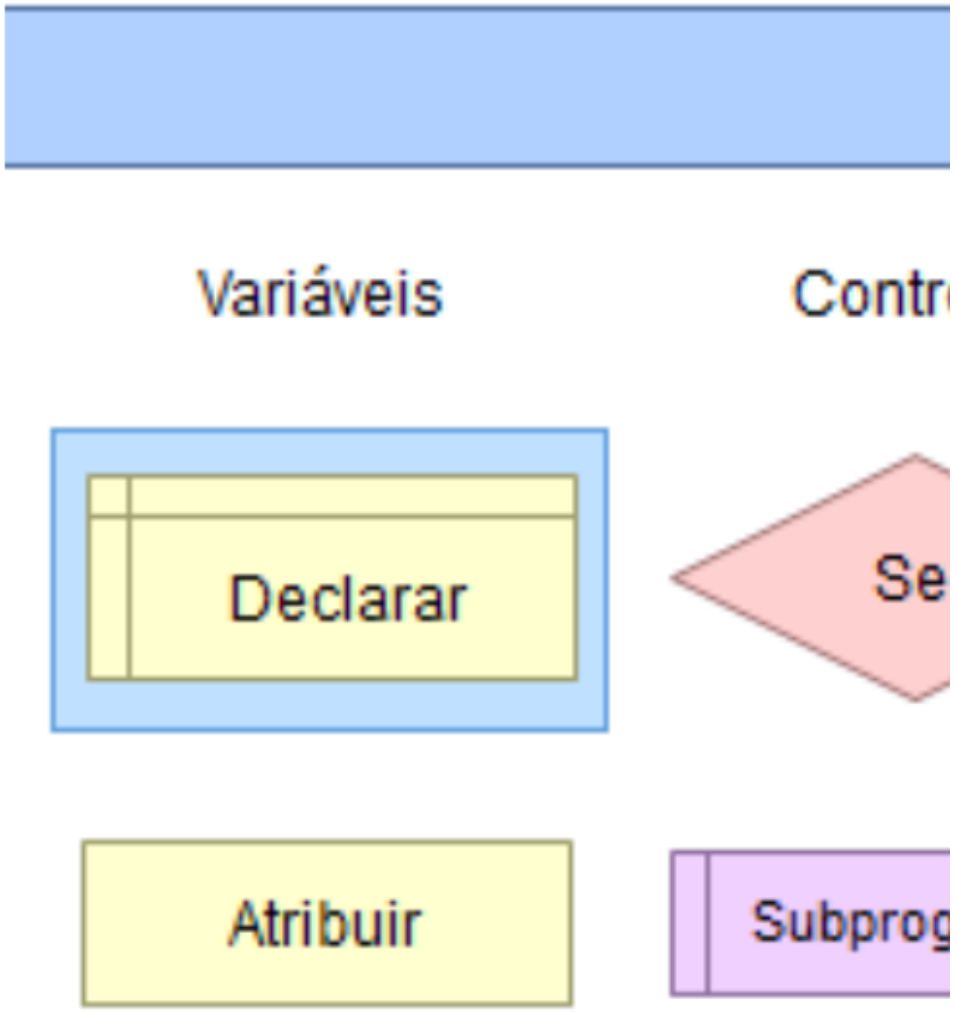
As variáveis permitem que o programa armazene e manipule dados, tornando-o mais dinâmico e interativo.

Declaração de Variáveis

- Antes de usar uma variável em C, é necessário declará-la, informando seu tipo e nome.
- A declaração reserva o espaço necessário na memória para a variável e a associa ao seu nome.

Exemplo de declaração de variável:

int idade; // Declaração de uma variável do tipo inteiro chamada "idade"



Atribuição de Valores

- Após declarar uma variável, podemos atribuir um valor a ela usando o operador de atribuição =.
- Isso armazena o valor na memória associada à variável.

Exemplo de atribuição de valor:

```
idade = 25; // Atribui o valor 25 à variável "idade"
```

Também é possível realizar a declaração e atribuição de valores em uma única linha:

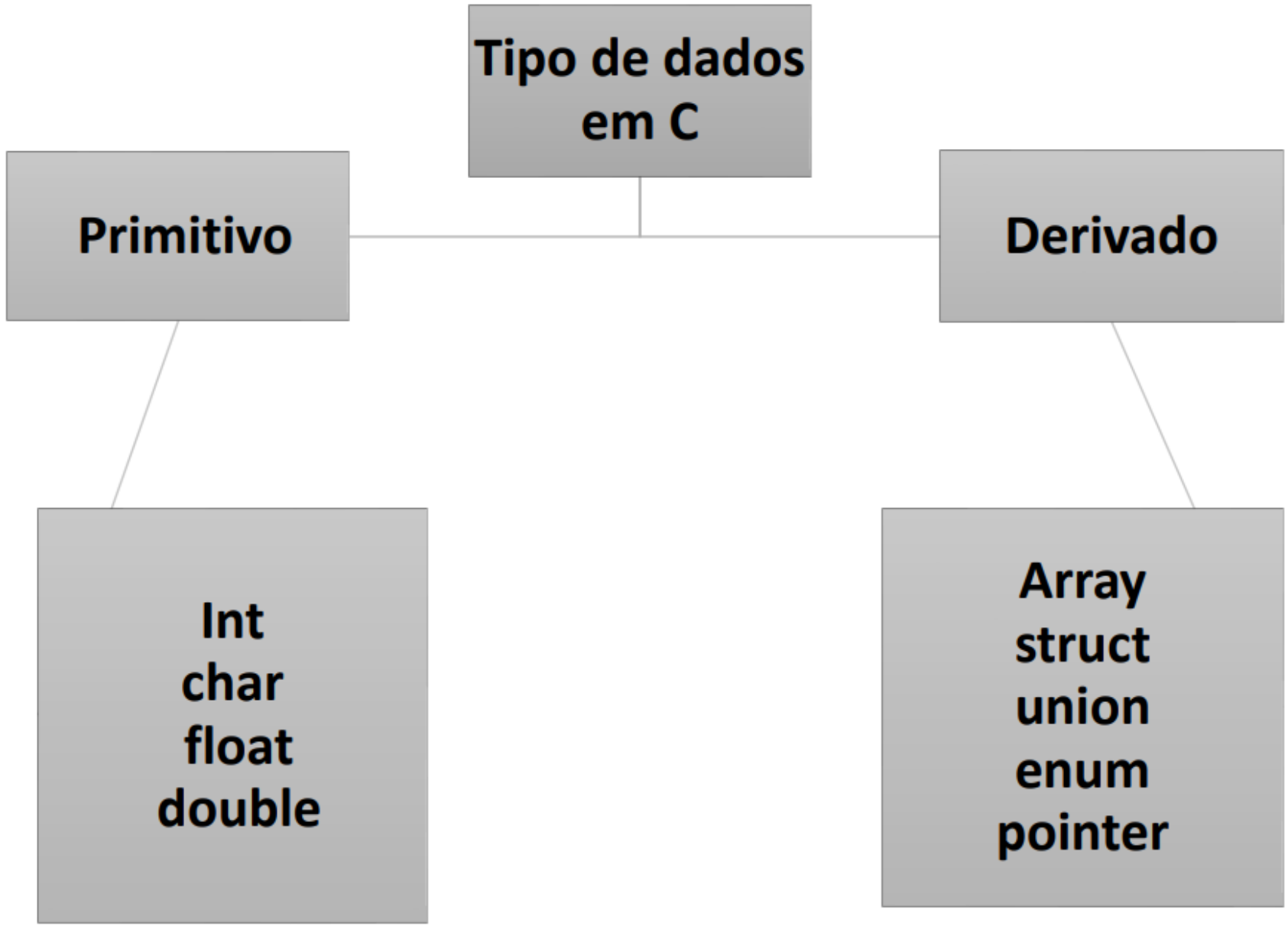
```
int idade = 25; // Declara e atribui o valor 25 à variável "idade"
```

Recomendação para o Identificador da Variável

- C é case sensitive. Recomenda-se começar com letras minúsculas.
- Não podendo começar com caracteres especiais ou números.
- Até pode iniciar com o caractere _ (underline), mas prefira utilizá-lo como separador de palavras.

- Identificadores desejáveis: endereco, valorDeCompra, valor_de_compra.
- Identificadores indesejáveis e/ou incorretos: Endereco, endereço, 12pessoas, _valordecompra

Tipos de Dados Básicos em C



Tipos de Dados Básicos em C

C possui vários tipos de dados básicos que podem ser usados para declarar variáveis. Alguns dos principais tipos de dados são:

- **int**: Representa números inteiros, como -5, 0, 42, etc.
- **float**: Representa números de ponto flutuante (com casas decimais), como 3.14, -0.5, etc.
- **char**: Representa caracteres individuais, como 'A', 'b', '7', etc.
- **double**: Similar ao float, mas com maior precisão.
- **void**: É usado para indicar a ausência de tipo. Por exemplo, uma função com retorno void não retorna nenhum valor.

O que significa retornar valor?

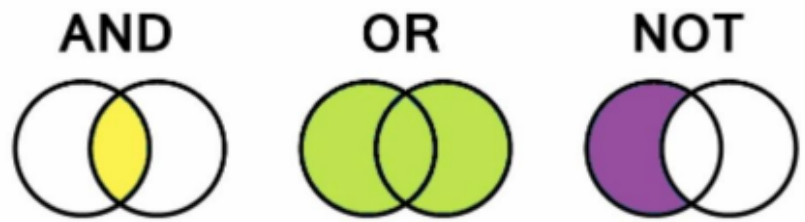
Significa retornar algo para quem chamou. Por exemplo:

double v1 = mediaAritmética(8, 10);

Observe como a função **mediaAritmética(double, double)** retorna um valor para a variável **v1**.

Booleano

- O tipo booleano que é suportado em diversas linguagens não existe no C.
- As variáveis booleanas armazenam dois tipos de dados: true ou false.
- Por serem valores de extrema importância, eles podem ser emulados em C por meio de uma variável do tipo inteiro que assume o valor 1 para verdadeiro e 0 para falso.



Modificadores de Tipos de Dados

- Modificam as tipagens padrão ‘char’ e ‘int’
- Short: reduz a capacidade de armazenamento e a faixa de representação → economia de memória
- Long: aumenta a capacidade de armazenamento, útil quando é necessário representar valores que excedem a capacidade padrão do tipo.
- Signed: é implicitamente assumido para tipos inteiros, indicando que podem representar valores positivos e negativos.
- unsigned é usado para tipos inteiros, indicando que devem representar apenas valores não negativos, ampliando a faixa de representação positiva. Útil quando apenas valores não negativos são necessários.

Variáveis Locais e Globais

- As variáveis globais são acessíveis a todos os escopos (trechos) do programa. São declaradas no início do código, antes da função main. É interessante evitar seu uso porque a partir de qualquer ponto esse valor pode ser modificado. Isso dificulta o controle e a leitura do código. Além disso, essas variáveis ocupam espaço de memória ao longo de todo programa, só sendo desocupadas ao final da execução.
- As variáveis locais são restritas a uso apenas nos escopos em que foram declaradas. Ocupam espaço na memória apenas enquanto os seus respectivos trechos estiverem em execução.

Exemplo

```c
#include <stdio.h>

int a; // variavel global (acessivel em todo o programa)

int main()
{
    int b; // variavel local (acessivel so na funcao main)

    return 0;
}
```

Constantes

- Além das variáveis, é possível usar constantes em C para representar valores fixos que não podem ser alterados durante a execução do programa.
- Existem 3 comandos diferentes que permitem o uso de constantes em C: **const**, **define** e **enumerate**.

Constantes com const

Podemos transformar variáveis em constantes adicionando o qualificador const no início da declaração da variável

Exemplo com o comando **const**:

```
const int diasPorSemana = 7; // Declara uma constante chamada "diasPorSemana" com o valor 7.
```

Constantes com define

Com o comando define, definimos as constantes no início do programa, logo abaixo das bibliotecas e antes do programa principal. Sugere-se que constantes tenham sempre letras maiúsculas.

Exemplo com o comando **define**:

```
#define NOME_DA_CONSTANTE valor_da_constante
```

```
#include <stdio.h>
#define MAX_ALUNOS 20
```

Importância da escolha do tipo de dado

- É importante escolher o tipo de dado correto para as variáveis, levando em conta o intervalo de valores que elas precisam armazenar e a quantidade de memória necessária.
- O uso adequado de tipos de dados ajuda a evitar erros e economizar recursos de memória.

Entrada e Saída de Dados

Função printf(): Saída de Dados

A função **printf()** é usada para imprimir dados na tela, por exemplo:

```
#include <stdio.h>

int main() {
    int idade = 25;
    printf("Minha idade é %d anos.\n", idade);
    return 0;
}
```

Função printf(): Saída de Dados

Dependendo do tipo de dado que será exibido, utilizamos o padrão tipo correspondente:

tipoK	tipo de dado correspondente
%d	int
%f	float
%lf	double
%c	char

Exemplo

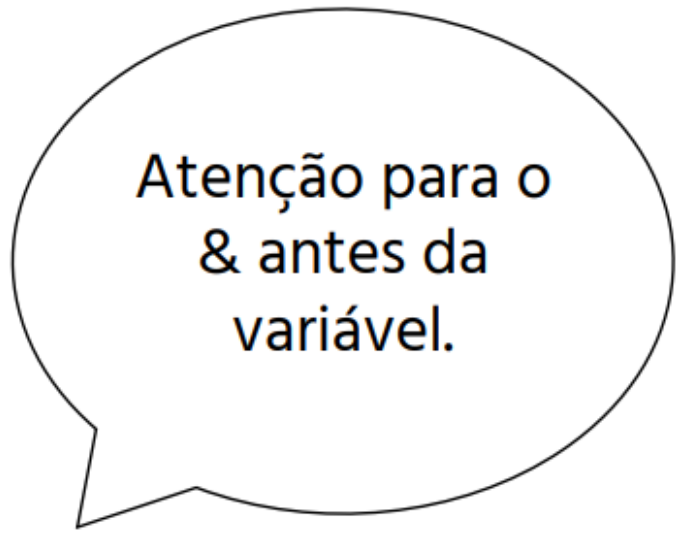
```
5   // declaracoes
6   int idade = 26;
7   float valorDo Pgto = 12.42;
8   double velParticula = 2.92817029837;
9   char tipoHabMotor = 'A';
10
11   // instrucoes
12   printf("Idade: %d\n", idade);
13   printf("Valor do pagamento : %f \n", valorDo Pgto);
14   printf("Velocidade da particula: %l f \n", velParticula);
15   printf("Tipo de Habilitacao: %c", tipoHabMotor);
```

Função scanf(): Entrada de Dados

A função scanf() é usada para capturar dados digitados pelo usuário, por exemplo:

```
#include <stdio.h>

int main() {
    int numero;
    printf("Digite um número inteiro: ");
    scanf("%d", &numero);
    printf("Você digitou o número %d.\n", numero);
    return 0;
}
```



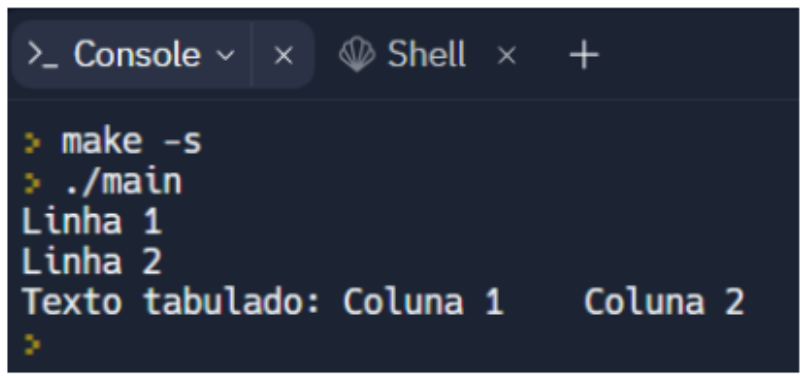
Caracteres de Escape

O \n insere uma quebra para uma nova linha e \t insere o espaço de tabulação. Ambos são usados para formatar a saída de texto.

Exemplo:

```c
#include <stdio.h>

int main() {
    printf("Linha 1\nLinha 2\n");
    printf("Texto tabulado:\tColuna 1\tColuna 2\n");
    return 0;
}
```



Expressões e Operadores Aritméticos

operador	formato	função
+	$e_1 + e_2$	soma e_1 com e_2
-	$e_1 - e_2$	subtrai e_1 de e_2
*	$e_1 * e_2$	multiplica e_1 por e_2
/	e_1 / e_2	divide e_1 por e_2
%	$e_1 \% e_2$	resto da divisão inteira de e_1 por e_2

Operadores de Atribuição

operador	formato	função
++	e_1++	$e_1 = e_1 + 1$
--	e_1 - -	$e_1 = e_1 - 1$
+=	e_1 += e_2	$e_1 = e_1 + e_2$
-=	e_1 -= e_2	$e_1 = e_1 - e_2$
*=	e_1 *= e_2	$e_1 = e_1 * e_2$
/=	e_1 /= e_2	$e_1 = e_1 / e_2$
%=	e_1 %= e_2	$e_1 = e_1 \% e_2$

Ordem de Precedência

Precedência de operadores diz respeito à forma como a linguagem C executa as expressões e seus operadores.

Ordem de precedência	Operadores
1°	( )
2°	* / %
3°	+ –
4°	=

Exercício 1: Calculadora Simples

Peça ao usuário para inserir dois números inteiros e realize as quatro operações básicas (adição, subtração, multiplicação e divisão) com esses números. Imprima os resultados na tela.

Exemplo de Saída:

Digite o primeiro número: 10

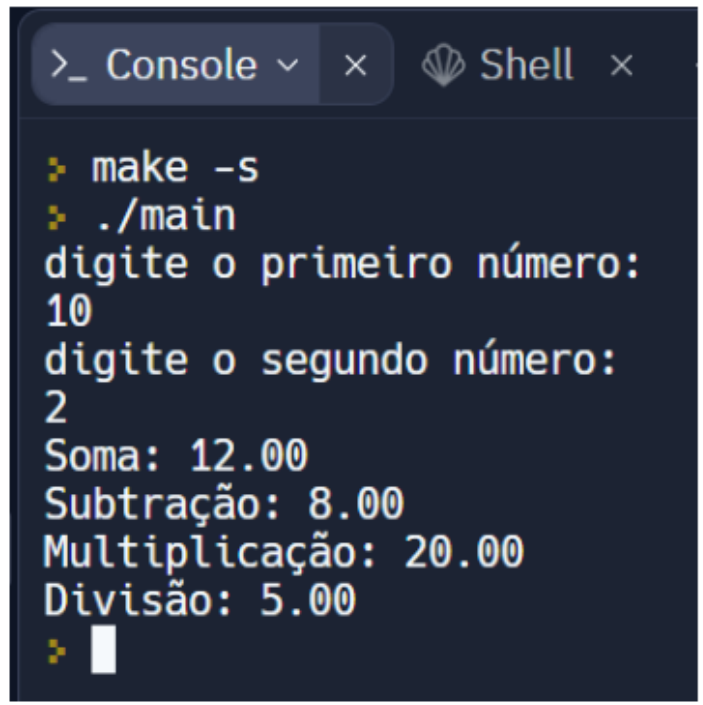
Digite o segundo número: 5

Soma: 15

Subtração: 5

Multiplicação: 50

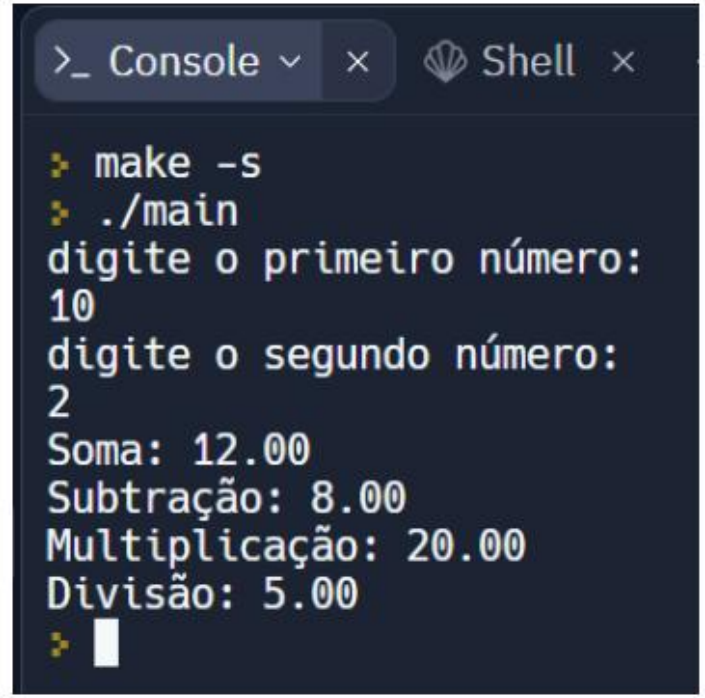
Divisão: 2



Exercício 1 - Resposta

```c
#include <stdio.h>

int main(void) {
    float n1;
    float n2;
    printf("digite o primeiro número:\n");
    scanf("%f", &n1);
    printf("digite o segundo número:\n");
    scanf("%f", &n2);
    float soma = n1 + n2;
    printf("Soma: %.2f\n", soma);
    printf("Subtração: %.2f\n", (n1-n2) );
    printf("Multiplicação: %.2f\n", (n1*n2));
    printf("Divisão: %.2f\n", (n1/n2));
    return 0;
}
```



Exercício 2: Conversão de Temperatura

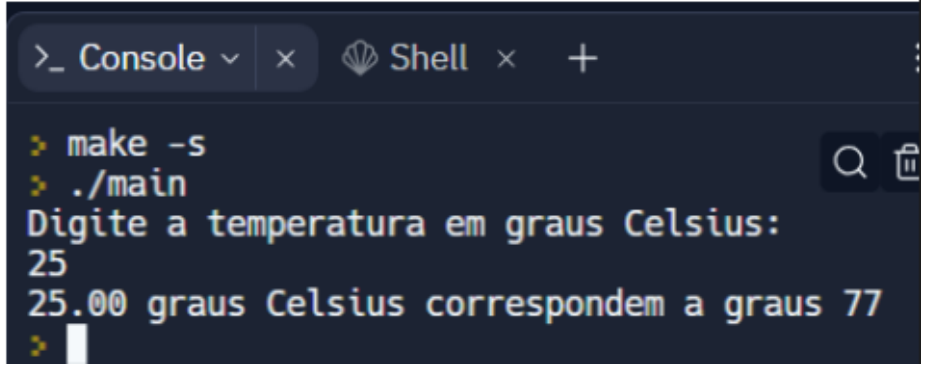
Peça ao usuário para inserir uma temperatura em graus Celsius e converta-a para graus Fahrenheit. A fórmula de conversão é:

F = (C * 9/5) + 32. Imprima o resultado na tela.

Exemplo de Saída

Digite a temperatura em graus Celsius: 25

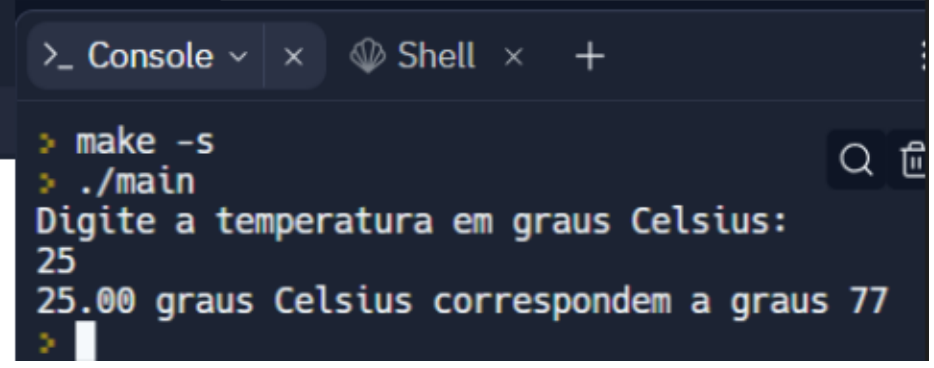
25 graus Celsius correspondem a 77 graus Fahrenheit.



Exercício 2 - Resposta

```c
#include <stdio.h>

int main(void) {
  float celsius;
  float farenheit;
  printf("Digite a temperatura em graus Celsius: \n");
  scanf("%f", &celsius);
  farenheit = ((celsius * 9/5) + 32) ;
  printf("%.2f graus Celsius correspondem a graus %.2f Farenheit", celsius  , farenheit);
  return 0;
}
```



Exercício 3: Cálculo de Média

Peça ao usuário para inserir três notas de um aluno (cada nota de 0 a 10) e calcule a média aritmética das notas. Imprima a média na tela.

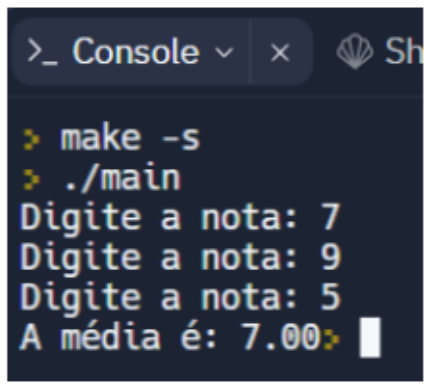
Exemplo de Saída

Digite a primeira nota: 8

Digite a segunda nota: 7

Digite a terceira nota: 6

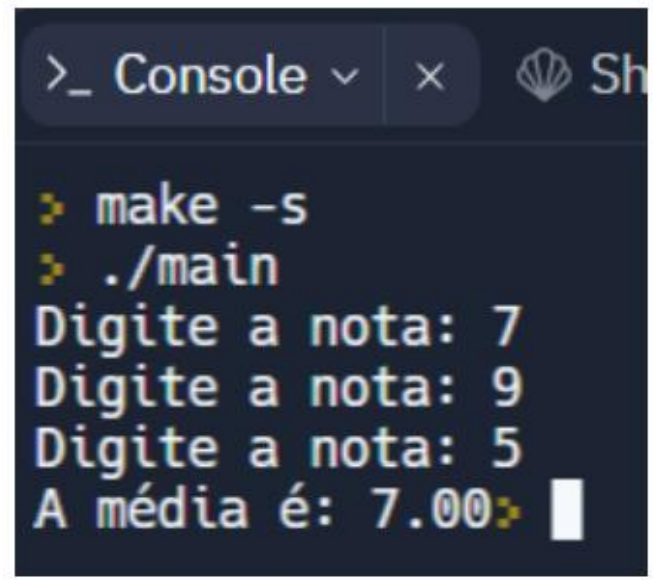
A média das notas é: 7.0



Exercício 3 - Resposta

```c
#include <stdio.h>

int main(void) {
    float n1, n2, n3, media;
    printf("Digite a nota: ");
    scanf("%f", &n1);
    printf("Digite a nota: ");
    scanf("%f", &n2);
    printf("Digite a nota: ");
    scanf("%f", &n3);
    media = (n1+n2+n3)/3;
    printf("A média é: %.2f", media);
    return 0;
}
```



Exercício 4: Troca de Valores

Peça ao usuário para inserir dois valores inteiros e troque os valores das variáveis sem a utilização de uma terceira variável auxiliar. Em seguida, imprima os novos valores.

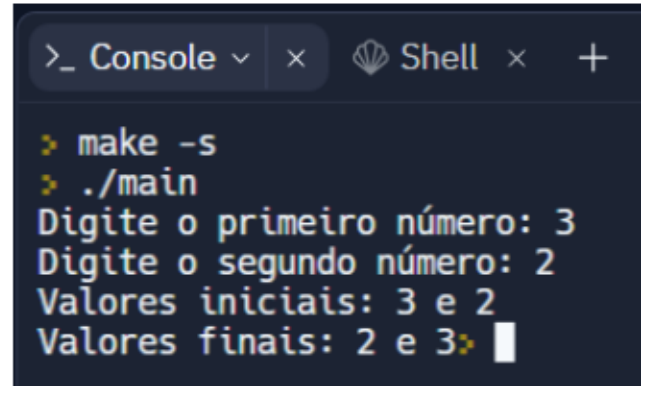
Exemplo de Saída

Digite o valor da variável A: 5

Digite o valor da variável B: 10

Valores antes da troca: A = 5, B = 10

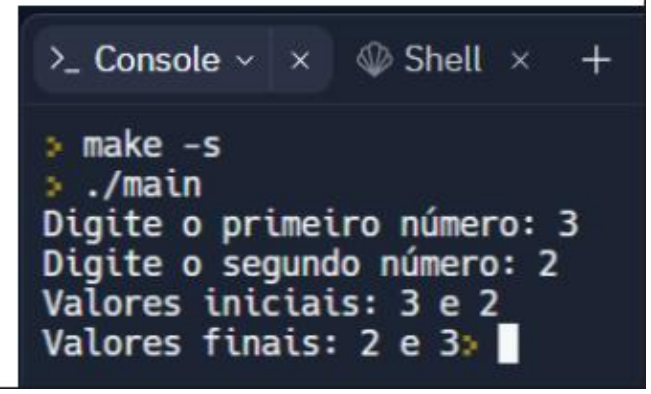
Valores depois da troca: A = 10, B = 5



Exercício 4 Resposta

```c
#include <stdio.h>

int main(void) {
  int a, b;
  printf("Digite o primeiro número: ");
  scanf("%d", &a);
  printf("Digite o segundo número: ");
  scanf("%d", &b);
  printf("Valores iniciais: %d e %d\n", a , b);
  a = a + b;
  b = a - b;
  a = a - b;
  printf("Valores finais: %d e %d", a , b);
  return 0;
}
```



Encontre o erro no código

Exercício 5

Objetivo?

Entradas?

Saídas?

```
8 #include <stdio.h>
9
10 int main()
11 {
12     int a = 1 , float b = 3;
13
14     printf("a = %d , b = %f \n", a , b);
15
16     return 0;
17 }
```

Exercício 5

Só podemos declarar múltiplas variáveis numa mesma linha separadas por vírgula, quando forem do mesmo tipo.

A correção, seria trocar a vírgula, por um ponto e vírgula na linha 12.

```
8 #include <stdio.h>
9
10 int main()
11 {
12     int a = 1 , float b = 3;
13
14     printf("a = %d , b = %f \n", a , b);
15
16     return 0;
17 }
```

Exercício 6

Objetivo?

Entradas?

Saídas?

```
8 #include <stdio.h>
9
10 int main( )
11 {
12     int a = 1;
13
14     printf("Digite o valor:");
15
16     scanf("%d", a);
17
18     printf("a = %d\n", a);
19
20     return 0;
21 }
```

Exercício 6

Faltou o & na linha 16.

```
8 #include <stdio.h>
9
10 int main( )
11 {
12     int a = 1;
13
14     printf("Digite o valor:");
15
16     scanf("%d", a);
17
18     printf("a = %d\n", a);
19
20     return 0;
21 }
```

Exercício 7

Objetivo?

Entradas?

Saídas?

```
8 #include <stdio.h>
9
10 int main()
11 {
12     int 1 a = 5 , b2 = –2;
13
14     printf("Valores 1a e b2: %d %d\n", 1a, b2);
15
16     return 0;
17 }
```

Exercício 7

Não podemos utilizar números para iniciar o identificador da variável.

Erro na linha 12

```
8 #include <stdio.h>
9
10 int main()
11 {
12     int 1 a = 5 , b2 = -2;
13
14     printf("Valores 1a e b2: %d %d\n", 1a, b2);
15
16     return 0;
17 }
```

CURSO SUPERIOR DE ADS

Apresentação da Disciplina e Paradigmas de Programação

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS