# Programação Imperativa e Funcional

## Apresentação e Introdução

Diego Bezerra

**dfb2@cesar.school**

C.E.S.A.R school

# Objetivos da disciplina

- Introduzir **conceitos** de **paradigmas de programação** e explorar outras linguagens de programação
- **Para alcançar esses objetivos...**
  - **Estudaremos os fundamentos dos paradigmas funcional e imperativo** utilizando uma linguagem de programação como referência (C e Haskell)
  - **Resolveremos (muitos) problemas práticos** utilizando C.

# Pré-requisitos

- Tem como pré-requisito a disciplina de **Fundamentos de programação:** estruturas de decisão, repetição, variáveis, tratamento de erros e outros conceitos necessários para a resolução de problemas
- PIF também é pré-requisito para outras 3 disciplinas: **Algoritmos e Estruturas de Dados**, **Infraestrutura de Hardware (3º período)** e **Infraestrutura de Software (4º período)**
    - Programação de mais baixo nível, gerenciamento de memória, operações e estruturas mais complexas e novos meios de pensar programação

# Sobre a disciplina

- **Aulas:** quinta-feira das 19h às 22h
- **Intervalos** de 15 minutos às 20h45
- **Duas chamadas:** às 19h25 e às 21h (logo após o intervalo)
- Sala do Google Classroom: **a2g7vmg**
- Canal no Slack: **#pif-adsb-20251**
  - Para mensagens urgentes!
- **É responsabilidade de todos acompanharem o classroom e slack constantemente**

# Conteúdo programático

- **Unidade 1:** Conceitos básicos da linguagem C, estruturas de controle, **vetores**, cadeias de caracteres, funções e procedimentos e **recursividade**
- **Unidade 2: Tipos estruturados, ponteiros, alocação dinâmica de memória** e conceitos básicos do paradigma funcional utilizando Haskell: listas e tuplas, tipos de dados e **polimorfismo**, **funções de alta ordem e recursão**

# Avaliações

- **Três tipos de avaliações: listas de exercícios semanais, provas e projetos**
  - **Unidade 1**: Listas de exercícios semanais (30%) e provas (70%)
  - **Unidade 2**: Listas de exercícios semanais (30%) e Projeto (70%)
- **Atividades** e **desafios** lançados **durante as aulas** podem render uma **pontuação extra!**

# Listas de exercícios

- As **listas de exercícios** consistem em duas ou três questões práticas de nível básico à avançado cobrindo o assunto visto naquela semana. O prazo para entrega é de 7 dias.
- Submissão via Beecrowd
  - **Criem uma conta em beecrowd!**
    - **https://judge.beecrowd.com/pt/login**
  - Depois acessem a turma do Beecrowd: bcwd.me/d-13954
  - Chave: **hTRjbrH**

# Listas de exercícios

- Regras de submissão: questões devem ser submetidas no prazo tanto pelo **Beecrowd quanto pelo Classroom**
- Pelo classroom:
  - Nomeie cada questão (arquivo.c) de sua lista como qn_login.c, onde n é o número da questão e login é seu e-mail da school
    - Exemplo: Diego vai submeter a questão 1, logo q1_dfb2.c
  - Coloque todas as questões em uma pasta nomeada com seu login (dfb2 seguindo o exemplo anterior)
  - Transforme a pasta em uma pasta zipada (.zip). Outro método não será aceito!
  - Submeta o arquivo .zip na atividade do classroom correspondente
- **Submissões fora do formato resultam em penalidade!**

# Provas

- As **provas** consistem em **até cinco questões teóricas e práticas** feitas em sala de aula com **papel e caneta**. A duração média é de 1h.
- Tem por objetivo consolidar o conhecimento visto em sala e praticado nas listas de exercícios
- Alunos com laudo médico possuem tempo extra para realização das atividades
  - 3 dias a mais para entregar as listas
  - 30 minutos a mais para a prova
  - Tais considerações só são válidas a partir do momento que o laudo é entregue ao apoio psicopedagógico

# Projeto

- **O projeto é um trabalho em grupo** a ser entregue no final da segunda unidade
  - A nota deste projeto será composta de **submissão dos artefatos gerados e a apresentação**.
  - Todos os membros devem apresentar e responder aos questionamentos feitos pelo professor. **A nota será individualizada**
  - Os melhores projetos serão convidados a apresentar na Mostra TechDesign e a participação pode render pontuação extra

# Cronograma

- As datas para as listas são o prazo de entrega
- Atrasos resultam em penalidade de **20% por dia**
  - Se eu atrasei 3 dias, terei redução de 3x20 = 60%
  - Último dia de submissão aceitável é de 3 dias depois do deadline
- Erros na formatação da submissão resultam em **penalidade de 20%**
- **Listas com questões copiadas serão zeradas**
- **Não há segunda chamada**

| **Unidade 1** | | **Unidade 2** | |
|---|---|---|---|
| **Atividade** | **Data** | **Atividade** | **Data** |
| Lista 1 | 27/02 | Lista 7 | 08/04 |
| Lista 2 | 13/03 | **Entrega Projeto** | **22/05** |
| Lista 3 | 20/03 | Apresentações | 22/05 |
| **Prova 1** | **20/03** | Apresentações | 29/05 |
| Lista 4 | 27/03 | Tech Design | 31/05 |
| Lista 5 | 03/04 | **SC** | **18/06** |
| Lista 6 | 10/04 | **Final** | **27/06** |
| **Prova 2** | **17/04** | | |

# Responsabilidades com a disciplina

- Como desenvolvedores, **é responsabilidade de todos**
  - Debugar seu código e corrigir os erros
  - Testar seu projeto antes de submeter
  - Atentar ao formato de submissão e os prazos de entrega
  - Buscar ajuda com os monitores, professores e colegas em casos de dúvidas
  - Comparecer às aulas, provas e apresentação do projeto

# Responsabilidades com a disciplina

- Como desenvolvedores, vocês estão aqui para **aprender** e se tornarem líderes no mercado de computação. Desta forma, **façam o uso consciente das ferramentas de IA.**
- **Você pode usar IA:**
  - Para **gerar exercícios para estudar**
  - Para te ajudar a debugar seu código (é diferente de pedir que ela conserte)
  - **Para encontrar bons materiais de estudo**
  - **Para criar estratégias de estudos**
- **Você não pode usar IA:**
  - Para fazer as listas de exercícios e projetos por você (**plágio**)

# Monitoria

- Monitores estarão disponíveis para tirar dúvidas e aprofundar alguns assuntos
  - Encontros síncronos - a tarde e remoto (datas a confirmar)
  - Encontros assíncronos - contatem os monitores via slack nos horários indicados
  - Participações nas monitorias serão avaliadas pelos monitores e valem pontuação extra no final das unidades (0.25 décimos)

# Dicas

- Preparem-se para **aprender dois paradigmas** e **duas linguagens de programação** diferentes em um curto intervalo de tempo!
  - A linguagem C tem curva de aprendizado maior que em Python
- **Exercitem**! **Exercitar constantemente** ajuda na fixação do conteúdo
- **Participem ativamente da disciplina**! Tirem dúvidas com professores, colegas e monitores
- **Lembrem-se**: falta reprova!

# Sobre mim

- Diego Bezerra <dfb2@cesar.school>
- **Formação**
  - Bacharel e Mestre em Ciência da Computação (UFRPE)
  - Doutorando em Ciência da Computação (CIn/UFPE)
- **Experiência**
  - Professor da CESAR School - 2022 - Atual
  - Pesquisador P&D do Grupo de Pesquisa em Redes e Telecomunicações - UFPE
  - Interesses: Sistemas Adaptativos, Aprendizado de Máquina, 5G e Internet das Coisas
  - https://www.linkedin.com/in/diegodefb/

# E sobre vocês?

- Nome? Como prefere ser chamado(a)?
- Experiência com a área?
- Por que escolheu o curso?
- Expectativas para o curso?

# Por que estudar dois paradigmas diferentes?

- Habilidade de **escolher o paradigma ideal** para diferentes problemas e linguagens
  - **Seleção** de **técnicas** que **otimizam desempenho** e **recursos**
- **Facilita** a adaptação a **novas linguagens**
- Expande a forma de abordar e **resolver problemas complexos**
  - Cada solução exigirá esforço e **diferentes maneiras de pensar** logicamente a resolução do mesmo problema
  - **Legibilidade** e **organização** do código
  - Aumento da produtividade

# Paradigmas de Linguagens de Programação

- Existem diferentes tipos de paradigmas
  - **Imperativo (sequência, atribuição, estado) (C, Java, C++)**
  - Orientado a objeto (objeto, estado, mensagem) (Java, Python, C++)
  - Orientado a eventos (processo, comunicação) (Delphi)
  - **Funcional (função, aplicação, avaliação) (Haskell, Scala)**
  - Lógico (relação, dedução) (QLISP, Prolog)
- **Dependendo da linguagem** utilizada, **mais de um paradigma** pode ser usado

# Paradigma Imperativo

- Orientada a **ações**, uma **sequência de instruções** que manipulam valores de **variáveis** (leitura e atribuição)
- Conceito de **estado** (modelado por variáveis) e **ações** (comandos) que manipulam o estado
- Descrevem como um programa deve operar
- **Como já sabemos, estudaremos este paradigma utilizando a linguagem de programação C!**

# Paradigma Funcional

- Expressa a lógica de um programa sem descrever o fluxo de controle
- Descrevem o que um programa deve alcançar em termos do domínio do problema
- Operações são expressas em termos de **funções** e um programa consiste de uma **sequência de funções** recursiva de alta ordem
- **Como já sabemos, estudaremos este paradigma utilizando a linguagem de programação Haskell!**

# Sugestões

- **Livro:**
  - Rangel Netto, J. L. M., Cerqueira, R. D. G., & Celes Filho, W. (2004). **Introdução a estrutura de dados: com técnicas de programação em C**.
- **Tutoriais**
  - Free Interactive tutorial - https://www.learn-c.org/
  - Learning C programming - https://www.tutorialspoint.com/cprogramming/index.htm
  - W3School - https://www.w3schools.com/c/index.php
  - Geeks for Geeks - https://www.geeksforgeeks.org/c-programming-language/
  - Code Academy - https://www.codecademy.com/catalog/language/c

# Programação Imperativa e Funcional

## Conceitos básicos da linguagem C

Diego Bezerra

**dfb2@cesar.school**

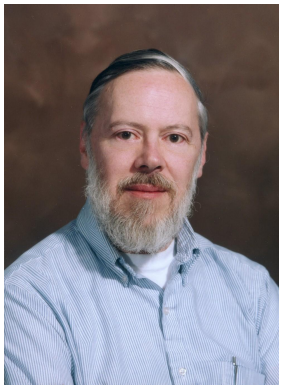
C.E.S.A.R school

# Objetivos da aula

- Iniciar na linguagem C
- Praticar o uso do Beecrowd

# História

- A linguagem C nasceu na década de 70 e atualmente é padronizada pela ANSI - American National Standards Institute
- C foi usado para desenvolver o sistema operacional UNIX
- Maior velocidade de processamento
    - Programas em C são compilados, gerando programas executáveis.
- Aplicação em diferentes arquiteturas



Dennis Ritchie

# Relembrando Algoritmos...

- Uma variável é um container, ela armazena algo
  - Espaço em memória
- As variáveis devem ter **tipo, tamanho e nome**
  - **Tipo**: Identifica a natureza da informação armazenada
  - **Tamanho**:
    - Capacidade de armazenamento da variável
    - Para algumas variáveis, o tamanho é definido pelo tipo
  - **Nome**: Identificador da variável, usado para acessá-la

# Tipos básicos em C

| Tipo | Valor | Tamanho (bits) | Valor mínimo | Valor máximo |
|---|---|---|---|---|
| **char** | Inteiro representa um caractere unicode | 8 | 0 ou -128 | 127 ou 255 |
| **int** | Inteiro | 16, 32 ou 64 | -32.768* | 32.767* |
| **float** | Oonto flutuante | 32 | 3.4E-38 | 3.4E+38 |
| **double** | Ponto flutuante | 64 | 1.7E-308 | 1.7E+308 |
| **void** | Sem valor | n/a | n/a | n/a |

# Tipos básicos e modificadores

| Modificador | Aplicável a | |
|---|---|---|
| **signed** | char, int | Permite valores negativos e positivos (padrão para int). |
| **unsigned** | char, int | Permite apenas valores positivos, dobrando o limite superior. |
| **short** | int | Reduz o tamanho do inteiro (geralmente 16 bits). |
| **long** | int, double | Aumenta o tamanho do tipo (long int geralmente 32 ou 64 bits, long double oferece maior precisão). |
| **long long** | int | Inteiro ainda maior que long (geralmente 64 bits, disponível a partir do C99). |
| **const** | Todos | Torna a variável somente leitura, impedindo modificações após a atribuição. |
| **volatile** | Todos | Indica que o valor da variável pode mudar inesperadamente (ex.: registradores de hardware, variáveis compartilhadas entre threads). |
| **restrict** | Ponteiros | Indica que um ponteiro é o único meio de acessar o objeto apontado (introduzido no C99, usado para otimizações). |

# Tipos básicos: declaração, inicialização e atribuição

```
int main() {
    //declaração de variável
    char d;
    float f;

    //declaração e inicialização de variável
    int x = 234;
    double m = 10.3, n=11.145;
    long long int t = 9223372036854775807LL;

    //inicialização de variável
    d = 'd';
    f = 32.5;

    return 0;
}
```

# Identificadores

- Identificadores não podem conter caractere especial, exceto **$** e **_**
  - Ex: **?ano, dia!mês** **INCORRETO**
  - Ex: **_ano, dia$mes** **CORRETO**
- Identificadores não podem começar com números
  - Ex. **1x, 1_t** **INCORRETO**
  - Ex. **x1, t_1** **CORRETO**
- Identificadores não podem conter operadores matemáticos
  - Ex: **dia+mes** **INCORRETO**

# Identificadores

- Palavras reservadas da linguagem não podem ser usadas como identificador

| | | | |
|---|---|---|---|
| auto | double | static | struct |
| break | else | int | switch |
| case | enum | long | typedef |
| char | extern | register | union |
| const | float | return | unsigned |
| continue | for | short | void |
| default | goto | signed | volatile |
| do | if | sizeof | while |

# Identificadores

- Identificadores são **case-sensitive**
- Convenção
  - Inicial minúscula
  - Junção de palavras capitalizadas
  - Nomes curtos e **significativos**

```c
int main() {
    int contador;
    int CONTADOR;
    int cOnTaDoR;
    return 0;
}
```

**Identificadores diferentes!**

```c
int main() {
    int ano;
    int anoNascimeto;
    return 0;
}
```

# Operadores aritméticos

- Aplicam-se a tipos numéricos (inteiro e ponto-flutuante)

| Operador | Operação |
|---|---|
| - | Menos unário (inversão de sinal) |
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão inteira |
| % | Resto da divisão inteira (módulo) |

```
int main() {
    int a = 5;
    int b = 3;
    int c = a % b;

    return 0;
}
```

# Operadores aritméticos

- Expressões são avaliadas da esquerda para a direita
- Por default, os operadores ***, /** e **%** possuem maior precedência do que **+** e **–**
  - É possível alterar a ordem usando parêntese
- Qual o resultado desta operação?

```
int y = 1 * 4 * 4 + 2 * 5 + 9;
```

# Operadores aritméticos de atribuição

- Aplicam-se a tipos numéricos (inteiros e ponto-flutuante)

| Operador | Equivalente a |
|---|---|
| a += b | a = a + b |
| a -= b | a = a - b |
| a *= b | a = a * b |
| a /= b | a = a / b |
| a %= b | a = a % b |

```
int main() {
    int a = 5;
    int b = 3;
    a += b; // a = a + b

    return 0;
}
```

# Operadores de comparação

- Também chamados de operadores relacionais
- Usados com valores/tipos numéricos, incluindo char
    - Comparação entre os valores Unicode correspondentes

| Operador | Valor | Aplicação | Resultado |
|---|---|---|---|
| **>** | maior do que | a > b | 1 se **a** é maior que **b**; 0, caso contrário |
| **>=** | maior do que ou igual a | a >= b | 1 se **a** é maior que ou igual a **b**; 0, caso contrário |
| **<** | menor do que | a < b | 1 se **a** é menor que **b**; 0, caso contrário |
| **<=** | menor do que ou igual a | a <= b | 1 se **a** é menor que ou igual a **b**; 0, caso contrário |

# Operadores de comparação

- Operadores de igualdade, um subtipo de operadores relacionais
- Podem ser usados com valores/tipos numéricos (incluindo char), booleanos e variáveis de referência

| Operador | Valor | Aplicação | Resultado |
|---|---|---|---|
| == | igual a | a == b | 1 se **a** é igual a **b**; 0, caso contrário |
| != | diferente de | a != b | 1 se **a** é diferente de **b**; 0, caso contrário |

# Operadores de lógicos

- Aplicam-se apenas à valores booleanos (expressões que resultem em true ou false)
- Operados booleanos

| Operador | Simbolo |
|---|---|
| Negação (**NOT**) | ! |
| Conjunção condicional (**AND**) | && |
| Disjunção condicional (**OR**) | \|\| |

# Entrada e saída básica

- As funções **printf** (saída) e **scanf** (entrada ) fazem parte da biblioteca **stdio.h**
  - Necessário importar a biblioteca usando **#include<stdio.h>**

```
#include<stdio.h>

int main() {

    return 0;
}
```

# Função printf

- A função **printf** possibilita a saída de valores e possui o seguinte formato:

```
#include<stdio.h>

int main() {
    printf(format, values);
    return 0;
}
```

| | |
|---|---|
| %c | especifica um char |
| %d | especifica um int |
| %u | especifica um unsigned int |
| %f | especifica um double (ou float) |
| %e | especifica um double (ou float) |
| %g | especifica um double (ou float) |
| %s | especifica uma cadeia de caracteres |

# Função printf

```
#include<stdio.h>

int main() {
    printf (“%d %g\n”, 33;5.3);
    // 33 5.3
    printf ("Inteiro = %d, Real = %g\n", 33, 5.3);
    // Inteiro = 33, Real = 5.3

    return 0;
}
```

# Função printf

- Formatando o número de casas decimais

```
#include<stdio.h>

int main() {
    printf (“%.2f\n”, 3.9087);
    // 3.91
    return 0;
}
```

# Função scanf

- A função scanf permite capturarmos valores fornecidos via teclado pelo usuário

```
#include<stdio.h>

int main() {
    scanf(format, values);
    return 0;
}
```

| | |
|---|---|
| %c | especifica um char |
| %d | especifica um int |
| %u | especifica um unsigned int |
| %f,%e,%g | especifica um float |
| %lf,%le,%lg | especifica um double |
| %s | especifica uma cadeia de caracteres |

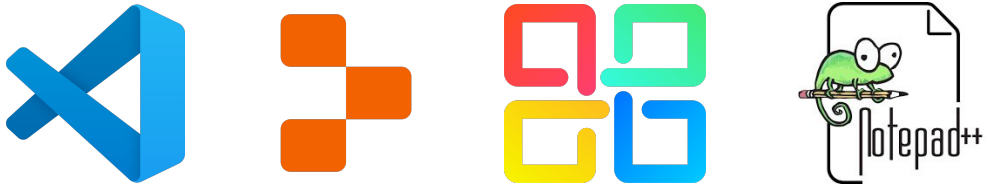
# Função scanf

```
#include<stdio.h>

int main() {
    int n;
    scanf(“%d”, &n);
    printf(“%d”, n);
    return 0;
}
```

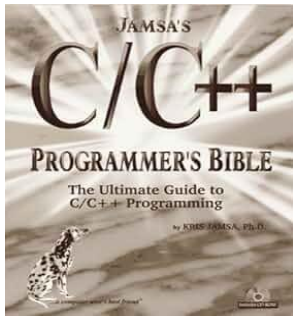
# Ferramentas para programar em C

- **Visual Studio (VS code)**: https://code.visualstudio.com/
- **Replit** (plataforma online): https://replit.com/languages/c
- **CodeBlocks**: https://www.codeblocks.org/
- **Notepad++**: https://notepad-plus-plus.org/



# Bibliografia


- JAMSA, Kris et KLANDER, L. A. R. S. **Programando em C/C++-a bíblia**. Ed. Makron Books, 1999.
- SCHILDT, H., **C Completo e Total**. Pearson, 1997.
- Thompson, S., Haskell: **The Crax Of Functional Programming.** Pearson; 2nd edition (April 8, 1999)
- DEITEL, Harvey M. et DEITEL, Paul J. **Como programar em C**. LTC, 1999.


C.e.S.A.R school