

# Paradigma Procedural

Linguagem C

Nome // RGM

Bruno de Alencar Ferreira // 30632391

Bruno Henrique Silva Nazaré // 29378443

Benjamin da Nóbrega Bezerra // 29274028

Gabriel Oliveira Faria // 29615267

Lívia Karla Tavares da Silva // 29682304

Pedro Felipe do Carmo Baptista // 30074924

# O que é Paradigma Procedural?

Tem esse nome pois se baseia em procedimentos que têm uma ordem determinada para ser executado, podendo ser feita pelo usuário ou por um sistema estabelecido e acrescentando **procedimentos** que organizem o código em blocos reutilizáveis.



## O que são procedimentos?

São sub-rotinas, possuem conjuntos de passos computacionais a serem executados. Podem ser chamados mutuamente, ou seja, um procedimento pode invocar outro para realizar uma subtarefa, tornando o código mais organizado e modular.



# Entendendo paradigma procedural

- A programação procedural, é derivada da programação imperativa, seu foco principal é que o programa pode ser dividido em blocos (funções), para serem chamados a qualquer momento, inclusive por outros procedimentos.
- Uma característica é a modularidade, que utiliza canais de entrada e saída estritamente definidos, usualmente acompanhados de regras claras sobre quais tipos de entrada e saída são esperados.
- Alguns pontos importantes de serem analisados serão descritos a seguir:





Deve suportar a especificação de tipos de argumentos, variáveis locais.



A linguagem precisa suportar a distinção do que é argumento de entrada e saída.



Alguns exemplos (ADA, Agl, C, Python(multiparadigma), Pascal), o foco desse relatório será a Linguagem de Programação C.

# Linguagem C

- Uma das mais influentes e utilizadas linguagens da história da computação;
- Desenvolvida para ser eficiente e simples, proporcionando a quem utiliza controle direto sobre os recursos;
- Tem foco na manipulação de memória, controle de fluxo e estrutura de dados;
- Amplamente empregada ao uso de desenvolvimento de sistemas operacionais, aplicações de baixo nível e compiladores.
- Apesar da estrutura simples, oferece suporte para uma ampla gama de aplicações, se tornando assim a base de muitas outras linguagens modernas;
- Focando na programação procedural, o código nessa linguagem é dividido em blocos, facilitando o reuso e a dinâmica como o algoritmo interage. Ponto bastante positivo da linguagem C.



# História da Linguagem C

# Nascimento da linguagem C

- Foi criada em 1972 por Dennis Ritchie, um cientista da computação americano, na empresa AT&T Bell Labs (Laboratórios Bell).
- Criada como uma evolução da linguagem B, que se deriva de BCPL (Basic Combined Programming Language).
- Tinha como objetivo fornecer uma linguagem de programação eficiente para o desenvolvimento do sistema operacional Unix

# Nascimento da linguagem C

- A principal motivação para a criação, foi a necessidade de ambientes portáteis e eficientes para implementar sistemas operacionais.
- Foi projetada para ser maleável e permitir o desenvolvimento de programas eficientes em termo de uso de memória e desempenho.
- Tradicionalmente, não possui um logotipo oficial amplamente reconhecido, mas o símbolo mais comum associado à linguagem é uma letra "C" em tipografia simples.



# Criadores da linguagem

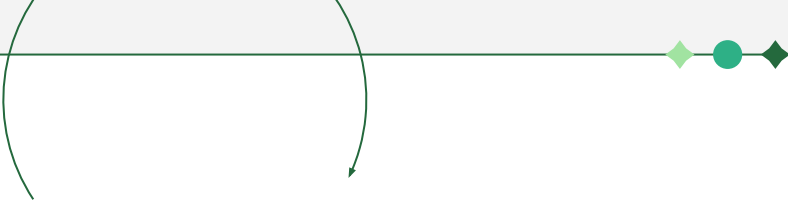
O principal desenvolvedor foi Dennis Ritchie, que trabalhou no desenvolvimento do Unix ao lado de Ken Thompson, que desenvolveu a linguagem B, antecessor direto de C.

Além destes, outros contribuintes como Alan Syder, Steven C. Johnson e Michael Lesk, tiveram seu papel para o aprimoramento da linguagem ao longo dos anos.



# Criadores da linguagem

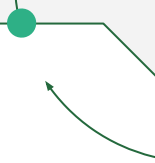
Em 1978, Dennis Ritchie e Brian Kernighan lançaram um livro intitulado "The C programming Language", que serviu como documentação e auxiliava os usuários e entusiastas de C com tutoriais aplicar funções e manipular variáveis.

O livro foi chamado de "K&R C", e foi considerado uma parte básica da linguagem, futuras edições iriam abranger mais tópicos da linguagem como os tipos de dados "struct", "long int", e "unsigned int"



**Como o C trabalha com o  
Paradigma Procedural e  
Porque é usada nesse  
paradigma?**

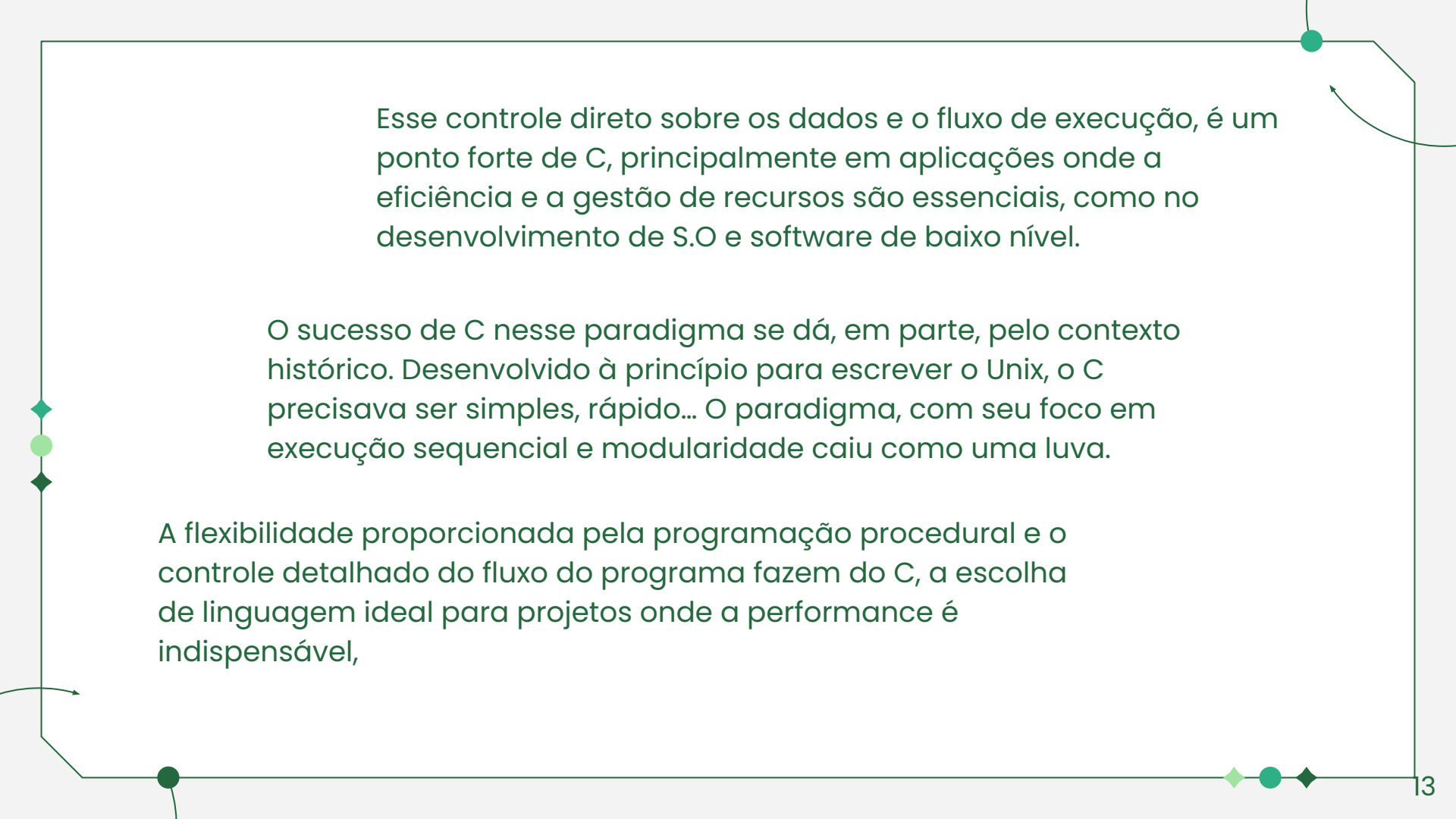




A linguagem segue o paradigma procedural, onde cada função é responsável por uma tarefa específica, e o fluxo do programa é controlado por chamadas a essas funções, permitindo que o código seja modular, reutilizável e fácil de manter.

Isso possibilita a quebra de problemas complexos, em partes menores, facilitando o desenvolvimento e a depuração do código. A modularidade desse paradigma, torna possível a organização do código de forma eficiente e compacta.

Em C, as funções podem receber parâmetros, processar dados e retornar resultados, oferecendo um fluxo claro de informações entre diferentes partes do programa. Isso é importante para que seja permitido ao programador decidir como os dados serão manipulados e armazenados.



Esse controle direto sobre os dados e o fluxo de execução, é um ponto forte de C, principalmente em aplicações onde a eficiência e a gestão de recursos são essenciais, como no desenvolvimento de S.O e software de baixo nível.

O sucesso de C nesse paradigma se dá, em parte, pelo contexto histórico. Desenvolvido à princípio para escrever o Unix, o C precisava ser simples, rápido... O paradigma, com seu foco em execução sequencial e modularidade caiu como uma luva.

A flexibilidade proporcionada pela programação procedural e o controle detalhado do fluxo do programa fazem do C, a escolha de linguagem ideal para projetos onde a performance é indispensável,



# Versões de C

# Linha do tempo das versões

→ C K&R: Essa versão do C é a que foi apresentada no livro "The C Programming Language", em que apresentava novos tipos de dados e algumas alterações na sintaxe, de "=" para "+=".

→ C99: Essa versão foi criada em 1999, graças a popularização do C++ e a falta de atualizações no C, foram adicionados tipos de dados novos, como o long int, e foram adicionadas algumas semelhanças ao C++, como o uso de "//" para comentários e remoção de restrições sobre a localização da declaração de variáveis.

# Linha do tempo das versões

→ **C11:** Em dezembro de 2011, essa versão foi lançada e o comitê de padrões de C resolveu adicionar um conjunto de regras para limitar a inserção de funcionalidades não testadas.

→ **C17:** Em 2017 essa versão teve como objetivo melhorar a revisão de funcionalidades não testadas, substituindo o C11.



# Características de C

- Simplicidade e eficiência
- Portabilidade
- Controle direto da memória



# Sintaxe e semântica da Linguagem C

# Sintaxe

São regras detalhadas para construção válida da linguagem C. Estas estão relacionadas com os tipos, declarações, funções e expressões. Os tipos definem as propriedades dos dados manipulados em um programa.

As declarações expressam as partes do programa, podendo dar significado a um identificador, alocar memória, definir conteúdo inicial, definir funções. As funções especificam as ações que um programa executa quando roda. A determinação e alteração de valores, e a chamada de funções de I/O são definidas nas expressões.

As funções são as entidades operacionais básicas dos programas em C, que por sua vez são a união de uma ou mais funções executando cada qual o seu trabalho. Há funções básicas que estão definidas na biblioteca C.

# Elementos da sintaxe em C

- Palavras-chave:

**Tabela 1.2** Uma lista das palavras-chave de C ANSI.

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

- Declarações e Expressões:

```
1 #include <stdio.h> // Biblioteca padrão para funções de entrada/saída
2
3 int main() {
4     // Declaração de variáveis
5     int a; // Declaração de uma variável inteira chamada 'a'
6     int b = 5; // Declaração e inicialização de uma variável inteira chamada 'b' com o valor 5
7     int resultado; // Declaração de uma variável inteira chamada 'resultado'
8
9     // Expressões
10    a = 10; // Atribuição de valor: a expressão "10" é atribuída à variável 'a'
11    resultado = a + b; // Expressão aritmética: soma de 'a' e 'b', atribuída à variável 'resultado'
12
13    // Impressão do resultado
14    printf("O resultado de a + b é: %d\n", resultado); // Saída do valor da variável 'resultado'
15
16    return 0; // Fim do programa
17 }
```



# Elementos da sintaxe em C

- Estrutura de Controle: IF

```
# include <stdio.h>

int main() {
    int numero;

    printf("Digite um número: ");
    scanf("%d", &numero);

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

    return 0;
}
```

- Estrutura de Controle: IF – ELSE – IF

```
# include <stdlib.h>
# include <time.h>

int main(){
    int num,segredo;

    srand(time(NULL));
    segredo=rand()/100;
    printf("Qual e o numero: ");
    scanf("%d",&num);
    if (segredo==num){
        printf("Acertou!");
        printf("\nO numero e %d\n",segredo);
    }
    else if (segredo<num)
        printf("Errado, muito alto!\n");
    else
        printf("Errado, muito baixo!\n");
}
```



# Elementos da sintaxe em C

- Estrutura de Controle: Loop For

```
int main()
{
    int x;
    for(x=1;x<100;x++){
        printf("%d\n",x);
    }
}
```

- Estrutura de Controle: Switch

```
switch(variável) {
    case constante1:
        seqüência de comandos
        break;

    case constante2:
        seqüência de comandos break;
    default:
        seqüência de comandos
}
```



# Elementos da sintaxe em C

- Estrutura de Controle: While

```
int main() {  
    char ch;  
    while(ch!='a') ch=getchar();  
}
```

- Estrutura de Controle: Do While

```
int main() {  
    char ch;  
    printf("1. inclusão\n");  
    printf("2. alteração\n");  
    printf("3. exclusão\n");  
    printf(" Digite sua opção:");  
    do  
    {  
        ch=getchar();  
        switch(ch)  
        {  
            case '1':  
                printf("escolheu inclusao\n");  
                break;  
            case '2':  
                printf("escolheu alteracao\n");  
                break;  
            case '3':  
                printf("escolheu exclusao\n");  
                break;  
            case '4':  
                printf("sair\n");  
                }  
        }  
    while(ch!='1' && ch!='2' && ch!='3' && ch!='4');  
}
```



# Semântica

A semântica é complementar à sintaxe de programas de computador, que se preocupa em descrever as estruturas de uma linguagem de programação

É escrever o código com a estrutura correta para a execução sequencial do código da linguagem e obtendo o resultado final

Erros semânticos em C ocorrem quando o código está sintaticamente correto, mas as operações que ele realiza não fazem sentido lógico, resultando em comportamento indesejado ou imprevisível



# Definição de semântica

A semântica das variáveis em C refere-se ao significado e comportamento das variáveis no contexto do programa. Isso envolve como os valores são atribuídos, como são armazenados na memória e como são utilizados durante a execução do programa.



## Exemplo:

//EXEMPLO CORRETO

```
int main() {  
    int a = 5;  
    int b = 3;  
    int resultado = a + b; // Operação válida entre variáveis de mesmo tipo  
    printf("Resultado: %d\n", resultado); // Saída: Resultado: 8  
    return 0;  
}
```

// EXEMPLO INCORRETO

```
# include <stdio.h>  
  
int main() {  
    int a = 5;  
    float b = 3.5;  
    int resultado = a + b; // Operação incorreta, possível perda de dados  
    printf("Resultado: %d\n", resultado); // Saída: Resultado é truncado  
    return 0;  
}
```

# Semântica de C

- Tipos de Dados e Operações: Discuta a importância dos tipos de dados (inteiros, flutuantes, caracteres, etc.) e como a semântica está envolvida na forma como as operações são realizadas sobre esses tipos.
- Semântica de Funções: Aborda o conceito de passagem de parâmetros por valor e referência, retorno de funções e o uso de ponteiros, com exemplos práticos.
- Erros Semânticos: Apresente exemplos de erros semânticos, como acessar uma variável não inicializada, divisão por zero, etc.





## Diferença entre Sintaxe e Semântica

# Sintaxe

Refere-se às regras e estrutura do código-fonte. Erros de sintaxe ocorrem quando o código não segue as regras de escrita da linguagem, como omitir um ponto e vírgula ou usar palavras-chave incorretamente. Erros sintáticos são detectados pelo compilador ou interpretador antes da execução do programa.

# Semântica

Refere-se ao significado do código. Mesmo que um programa esteja sintaticamente correto, ele pode conter erros semânticos se o comportamento não fizer sentido lógico. Por exemplo, calcular a raiz quadrada de um número negativo sem tratamento de erro. Erros semânticos são mais difíceis de detectar porque o código é compilado, mas não funciona como esperado.

# Ferramentas de análise

→ Compiladores:



# Ferramentas de análise

Na análise léxica, o compilador verifica se o código contém apenas símbolos válidos na linguagem de programação.

Na análise sintática, o compilador verifica se as instruções do código seguem a gramática da linguagem. Se a sintaxe estiver incorreta, o compilador emite mensagens de erro que indicam o tipo e a localização do problema.

# Exemplo de código Sintaticamente e Semanticamente corretos:

```
// Função que calcula o fatorial de um número
# include <stdio.h>

int fatorial(int n) {
    if (n < 0) {
        return -1; // Retorna -1 para indicar um erro (fatorial de número negativo não é definido)
    }
    int resultado = 1;
    for (int i = 1; i <= n; i++) {
        resultado *= i; // Multiplica resultado por 'i'
    }
    return resultado;
}
```



# Exemplo de Código com Erros Sintáticos

```
# include <stdio.h>

int main() {
    int x = 5 // Erro: ponto e vírgula ausente
    printf("O valor de x é: %d\n", x);
    return 0;
}
```

# Exemplo de Código com Erros Semânticos

```
# include <stdio.h>

int main() {
    int a = 10, b = 0;
    int resultado = a / b; // Erro semântico: divisão por zero
    printf("Resultado: %d\n", resultado);
    return 0;
}
```

# Referências

1. C (linguagem de programação). In: WIKIPÉDIA, a enciclopédia livre. São Francisco: Wikimedia Foundation, 2023. Disponível em: [https://pt.wikipedia.org/wiki/C\\_\(linguagem\\_de\\_programa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/C_(linguagem_de_programa%C3%A7%C3%A3o)). Acesso em: Set/2024.
2. XXIV Jornada de Iniciação Científica e Cultural (JICC) – PDF Paper sobre a Linguagem C. Disponível em: <https://www.gov.br/cti/pt-br/publicacoes/producao-cientifica/jicc/xxiv-jicc-2022/pdf/jicc-2022-paper-8.pdf>. Acesso em: Set/2024.
3. RITCHIE, Dennis M. The Development of the C Language. Bell Labs. Disponível em: <https://www.bell-labs.com/history/>. Acesso em: Set/2024.
4. Bell Labs History. The C Programming Language. Disponível em: <https://www.bell-labs.com/history/>. Acesso em: Set/2024.

# Referências

5. KERNIGHAN, Brian W.; RITCHIE, Dennis M. The C Programming Language. Prentice Hall, 1978.
6. Programação procedural – Wikipédia, a enciclopédia livre. Disponível em: [https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o\\_procedural](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_procedural). Acesso em: Set/2024.
7. BINSELY, A. Introdução à linguagem C. Porto Alegre: UFRGS, 2006. Disponível em: <https://www.inf.ufrgs.br/~binsely/tutorialc.pdf>. Acesso em: Set/2024.
8. COUTO, P. P. Compiladores. Campinas: UNICAMP, 2006. Disponível em: <https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node37.html>. Acesso em: Set/2024.

# Referências

9. SOUZA, C. Linguagens de Programação: (...)conceitos de linguagens interpretadas e compiladas. Petrolina: UNIVASF, 2017. Disponível em: <http://www.univasf.edu.br/~criston.souza/algoritmos/arquivos/aula02.3.pdf>. Acesso em: Set/2024.
10. LAGO, S. Linguagem C. São Paulo: IME-USP, 2015. Disponível em: <https://www.ime.usp.br/~slago/slago-C.pdf>. Acesso em: Set/2024.
11. OLIVEIRA, L. S. C completo e total. Curitiba: UFPR, 2012. Disponível em: <https://www.inf.ufpr.br/lesoliveira/download/c-completo-total.pdf>. Acesso em: Set/2024.
12. PINHO, F. Histórico da Linguagem C. Porto Alegre: PUCRS, 2001. Disponível em: <https://www.inf.pucrs.br/~pinho/Laprol/Historico/Historico.htm>. Acesso em: 10 set. 2024.



# Obrigado!