UNIPÊ
Centro Universitário
de João Pessoa

Técnicas e Desenvolvimento de Algoritmos

Prof Me Tiago Emílio
tearaujo@unipe.edu.br

UNIPÊ
Centro Universitário
de João Pessoa

Linguagem C

“Características, Paradigmas e Dados Elementares”

C Programming

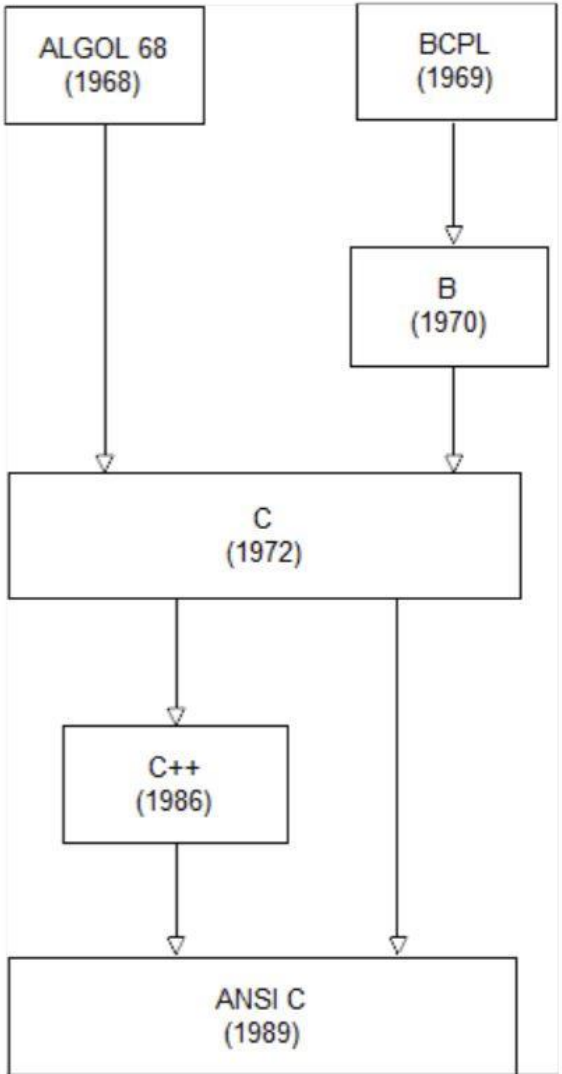
Linguagem considerada de **médio nível**:

Linguagens **baixo nível**: micro-arquitetura, assembly;

Linguagens de **médio nível**: C - Por quê?

Linguagem de **alto nível**: Linguagens interpretadas como Java, Python ou outras mais modernas como Go!

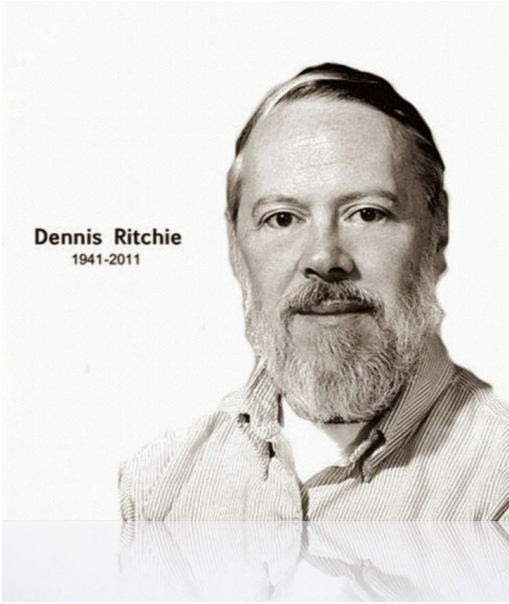
Surgiu para substituir a linguagem **Fortran**;



Em busca de se **isolar o programador** dos detalhes relativos a arquitetura do computador no qual se deseja rodar seu programa, buscou-se o desenvolvimento de **linguagens de alto nível**;

Linguagens como o Algol foram uma tentativa, apesar de frustrada;

Em meio termo temos, C, uma evolução de B e que foi criada para ser **rápida o suficiente** para **implementar o núcleo de um sistema operacional** e **simples o suficiente** para abstrair os **detalhes de arquitetura do programador;**



Dennis Ritchie - 1941 - 2011, foi o autor da linguagem.

Ele trabalhou por 40 anos na **Bell Labs**, ajudou a implementar o **Unix** e recebeu um dos maiores prêmios da área da computação, o **Prêmio Turing**, em 1984

Padrões

C é uma linguagem que se adequa a **qualquer propósito**, sendo **desassociada de arquitetura**;

É importante a toda linguagem de programação ser **portável** - ou seja, que a implementação de um programa seja levado a qualquer arquitetura sem dificuldades - C ANSI - 1988 e C99;

C é uma linguagem **compilada**, **estruturada** e **fortemente tipada**;

32 palavras reservadas;

Um programa em C é um texto cujo conteúdo segue a sintaxe da linguagem C:

```c
/*
    Comentário de Bloco
    Nada escrito aqui será interpretado
    pelo compilador.
*/
//  Comentário de linha
//  Nada nesta linha será interpretado pelo compilador

#include <stdio.h> // inclusão de bibliotecas

int main() // Declaração da função principal
{
    printf("Alo mundo!"); // Chamada de função
    return 0; // Fim da função principal, retornando inteiro
}
```

Para ser executado pelo computador, o programa acima deve primeiro ser **compilado**.

Um **compilador** para a linguagem C é um programa capaz de interpretar um programa em C, como o programa acima, e traduzí-lo para linguagem de máquina, gerando um **arquivo executável**.

Palavras reservadas

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

Estrutura de um programa C

```
inclusao bibliotecas

declarações globais

protótipos de funções

tipo devolvido main (lista de parâmetros) {
   seqüência de comandos
}

tipo devolvido f1(lista de parâmetros) {
   seqüência de comandos
}
...
tipo devolvido fN(lista de parâmetros) {
   seqüência de comandos
}
```

Expressões em linguagem C

Podemos classificar as expressões em dois grupos, as de **atribuição** e as de **comparação**;

As **expressões de atribuição** servem para inicializar variáveis.

A propósito, por que inicializar é importante?

int a = 10;

a = 10;

Variáveis locais são inicializadas a cada execução do seu escopo.

A única ressalva é com o modificador de acesso **static**, que trata a variável **mantendo seu valor entre chamadas**

Variáveis

Nome que se **refere** a um **valor**.

Um comando de atribuição **cria** uma nova variável e lhe dá um **valor**.

São usadas para **guardarmos valores** que serão usados mais tarde no programa.

Variáveis

Um **comando** é uma unidade de código que o interpretador pode executar.

Um comando de **atribuição** tem a seguinte forma:

```
variável = expressão
```

Cada nome de variável é uma letra ou o símbolo '_' (underscore) seguida de letras, números e '_'. **Nunca** pode começar com um número

Atenção, maiúscula é diferente de minúscula (**C é *case sensitive***).

Assim **maior** e **Maior** são duas variáveis diferentes.

Variáveis

```
variável = expressão
```

Ordem de execução:

1. o valor da expressão é calculado;
2. a variável é criada (alocada em memória);
3. a variável passa a fazer referência/guardar o valor em determinado endereço de memória.

Exemplos:

```
int num = 15;
int soma = 0;
soma = soma + num;
int a = 3;
int b = 4;
```

Constantes

Valores que por algum motivo devem aparecer em um programa:

```
int a = 3;
float pi = 3.14;
```

Tipos de variáveis

C possui alguns tipos básicos:

int: Corresponde aos números inteiros.

Ex: 10, -24.

float: Corresponde aos números racionais.

Ex: 2.4142, 3.14159265.

Há ainda outros tipos básicos:

bytes, listas (como texto, ou strings),

structs, unions

Tipos de variáveis em linguagem C

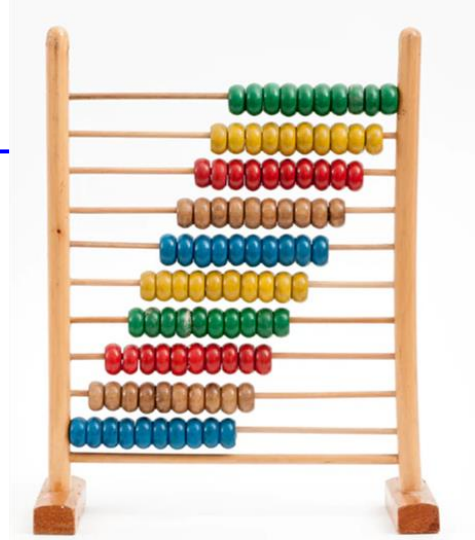
Atenção: atribuição funciona dentro de qualquer expressão válida:

Inclui até **IF** e **FOR**!

Vale ressaltar: apenas **variáveis** e **ponteiros** (variáveis especiais de endereços), podem receber valores;

Tipo do destino	Tipo da expressão	Possível informação perdida
signed char	char	Se valor > 127, o destino é negativo
char	short int	Os 8 bits mais significativos
char	int	Os 8 bits mais significativos
char	long int	Os 24 bits mais significativos
int	long int	Os 16 bits mais significativos
int	float	A parte fracionária e possivelmente mais
float	double	Precisão, o resultado é arredondado
double	long double	Precisão, o resultado é arredondado

Operadores aritméticos



+ Adição

- Subtração

***** Multiplicação

/ Divisão

% Mod (resto após divisão inteira)

E alguns outros:

--, ++;

Conhecem?

Os dois últimos são muito **eficientes** e devem ser utilizados **sempre que possível**.

Operadores relacionais

C **não possui** verdadeiro e falso como valores de uma variável;

Não existe o tipo boolean, segue a seguinte padronização

0: **falso**;

Qualquer valor diferente de 0: **verdadeiro**.

O resultado de uma expressão lógica verdadeira em C é **1**.

Operadores relacionais

Operador	Ação
>	Maior que
>=	Maior que ou igual
<	Menor que
<=	Menor que ou igual
==	Igual
!=	Diferente

Operadores lógicos

C **não possui** verdadeiro e falso como valores de uma variável;

Não existe o tipo boolean, segue a seguinte padronização

0: **falso**;

Qualquer valor diferente de 0: **verdadeiro**.

O resultado de uma expressão lógica verdadeira em C é **1**.

p	q	p&&q	p||q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Operadores lógicos

C **não possui** verdadeiro e falso como valores de uma variável;

Não existe o tipo boolean, segue a seguinte padronização

0: **falso**;

Qualquer valor diferente de 0: **verdadeiro**.

O resultado de uma expressão lógica verdadeira em C é **1**.

Operadores lógicos

Operador	Ação
&&	AND
! !	OR
!	NOT

Operadores bit a bit

Comparação de dois operandos bit a bit:

- AND: **&**
- OR inclusivo: **|**
- OR exclusivo: **^**

Deslocamento à esquerda: **<<**

Deslocamento à direita: **>>**

Complemento: **~**

Operadores bit a bit

AND	Os bits são definidos como 1 no resultado, se os bits correspondentes em ambos os operandos forem 1.
OR inclusivo	Os bits são definidos como 1 no resultado, se pelo menos um dos bits correspondentes em amobs os operandos for 1.
OR exclusivo	Os bits são definidos como 1 no resultado, se exatamente um dos bits correspondentes em ambos os operandos for 1.
Deslocamento à esquerda	Desloca os bits do primeiro operando à esquerda pelo número de bits especificado pelo segundo operando: preenche a partir da direita com zero (0) bits.
Deslocamento à direita	Desloca os bits do primeiro operando à direita pelo número de bits especificado pelo segundo operando: o método de preenchimento a partir da esquerda depende da máquina.
Complemento de um	Todos os bits 0 são definidos como 1, e todos os bits 1 são definidos como 0.

Operadores lógicos


```c
#include <stdio.h>
#include <stdlib.h>

int main(){

    int a = 10;
    int b = 10;
    unsigned char c;
    unsigned char d;
    unsigned char e;

    c = (a == b); //verifica se sao iguais
    d = (a == c); //verifica se sao iguais
    e = (a == c || a == b); //verifica se uma OU outra expressao sao iguais

    printf("O valor de C e: %d\n", c);
    printf("O valor de D e: %d\n", d);
    printf("O valor de E e: %d\n", e);

    return 0;
}
```

Desafio!

10 < 5 | | !(10 < 9) && 3 <= 4

Qual resultado?

E agora, qual resultado? **(10 < 5 | | !(10 < 9)) && 3 <= 4**

Desvios condicionais

Existem diversas formas de **controlar fluxo de execução** em função de sua regra de negócio implementada;

Desvios condicionais tornam possível esse controle e se utilizam de operadores lógicos para montar a estrutura de decisão desejada;

Vamos pensar!

Imagine um sistema embarcado em que a hora atual é verificada a cada 10s e testada para controlar uma lâmpada.

Como você implementaria a condição para acendimento da lâmpada? Vamos fazer o algoritmo!

Desvios condicionais

Em C temos 2 estruturas condicionais:

if: permite desvio de fluxo através de condicional simples ou composta, pode vir acompanhado de complemento **else**, mas é facultativo.

switch-case: substituto elegante para if, else aninhados em torno de um teste simples.

Através de uma comparação sob um tipo compatível com integer, efetua todos os testes de uma condição, executando instruções contidas na condição verdadeira e todas as demais condições abaixo.

Desvio condicional if

A estrutura lógica necessária para controlar a execução de um IF segue a mesma linha vista anteriormente, 0 é falso e diferente de 0, verdadeiro.

Veja uma condição das mais simples para ilustrar esse caso:

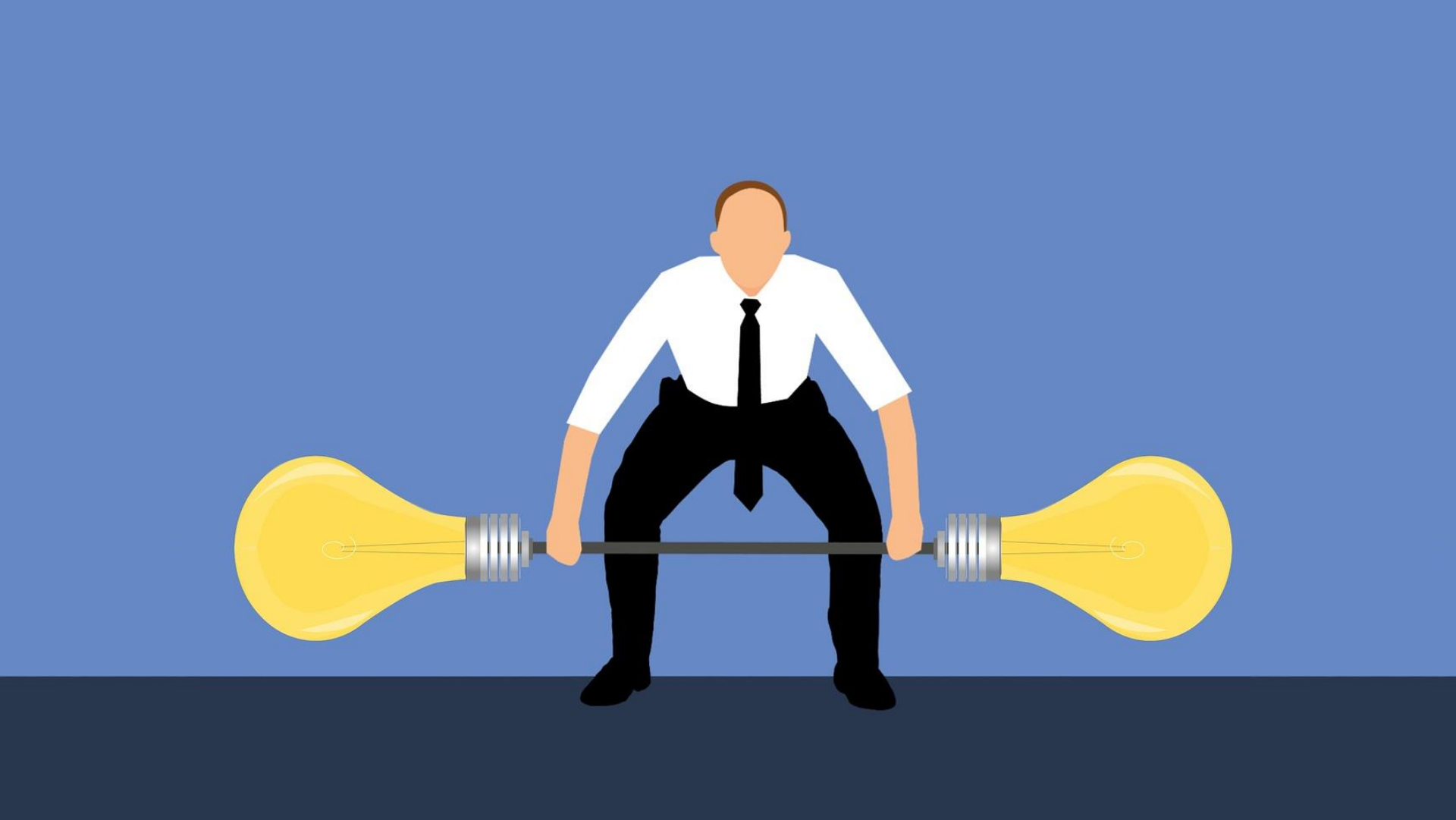
```c
#include <stdio.h>
#include <stdlib.h>

int main(){

    int a = 0;

    printf("Digite um numero para dividir 10\n");

    scanf("%d", &a);

    if(a)
        printf("%f\n", (float)10 / a);
    else
        printf("Voce digitou zero!\n");

    return 0;
}
```

Desvio condicional if e else

Vimos que a negativa, ou seja, a condição falsa também possua execução controlada utilizando-se do **else**;

Permite-se também que sejam **aninhados** diversos comandos, **if else**, formando uma “escada”:

```c
#include <stdio.h>
#include <stdlib.h>

int main(){

    int a = 0;

    printf("Digite um numero para dividir 10\n");

    scanf("%d", &a);

    if(!a){
        printf("Voce digitou zero!\n");
        return 1; //retorno de fracasso
    }
    else if(a < 0)
        printf("Voce digitou um numero abaixo de zero!\n");
    else if(a > 0)
        printf("Voce digitou um numero acima de zero!\n");

    printf("%f\n", (float)10 / a);

    return 0; //retorno de sucesso
}
```



Exercício

Escreva um programa no qual o **usuário** informe apenas a parte inteira da **hora.**

Para receber um valor inteiro digitado pelo usuário, utilize o **scanf:**

```c
#include <stdio.h>

int main()
{
    int numero;
    printf("Digite um numero inteiro: ");
    scanf("%d", &numero);
    printf("Numero digitado: %d", numero);
    return 0;
}
```

Exiba a mensagem de **bom dia**, **boa tarde** ou **boa noite** com **if aninhados**;

Para testar online, clique **aqui**.

Atividade para casa

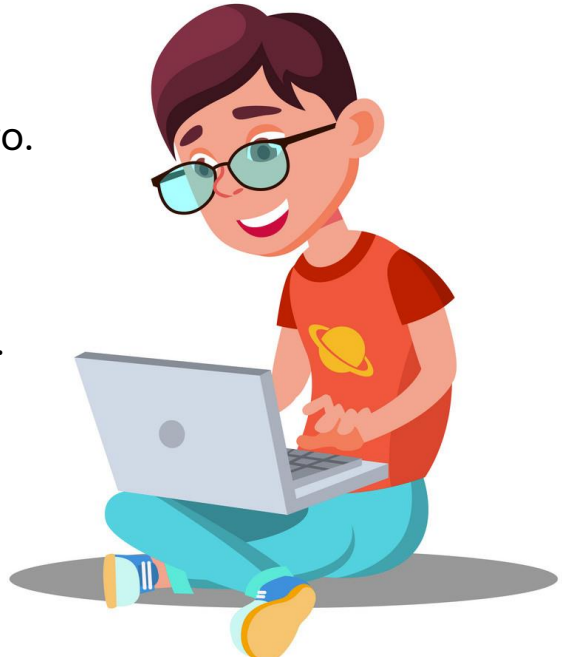
Solicite do usuário um número **inteiro de 1 a 12**.

Imprima o **mês** correspondente, sendo:

1 para Janeiro, 2 Fereveiro, até 12 para Dezembro.

Depois informe se o mês possui 31 dias ou não.

Utilize a menor quantidade de memória possível.



- DAMAS, Luiz Manoel D. **Linguagem C.** 10. ed. Rio de Janeiro: LTC Editora, 2007.

- SCHILDT, H. **C Completo e total.** São Paulo: Makron Books, 1976.

- DEITEL, Harvey M.; DEITEL, Paul J. **Como programar em C**. 6. ed. São Paulo: Livro Técnico e Científico, 2011.

- PINHO, Márcio S. Histórico da Linguagem C. Porto Alegue: PUC-RS, 2016. Disponível em: http://www.inf.pucrs.br/~pinho/LaproI/Historico/Historico.htm. Acesso em: 1 de agosto de 2018