Algoritmos e Programação de Computadores

Linguagem e Técnica de Programação - C

Programa da Disciplina

- **Organização básica de computadores**
- **Algoritmos**
- **Introdução a linguagem de programação estruturada**
- **Tipos de dados, identificadores, constantes e variáveis**
- **Comandos de entrada/saída e operadores**
- **Estruturas de decisão (*if*, *switch*)**
- **Estruturas de repetição (*for*, *while*, *do-while*)**
- **Avaliação P1**
- **Vetores e matrizes**
- ***Strings* e estruturas**
- **Modularização de programas**
- **Ponteiros e alocação dinâmica de memória**
- **Avaliação P2**

Sumário

<u>Introdução a linguagem de programação estruturada</u>

- **Transformação de algoritmo para programa**
- **Tipos de linguagem de programação (*linguagem de máquina, de baixo nível e de alto nível*)**
- **Paradigmas de linguagem de programação (*programação estruturada e orientada a objetos*)**
- **Compiladores e interpretadores**
- **A linguagem de programação C**
- **Primeiro programa em C**
- **Tipos de erros em programas (*erros de compilação, de linkedição e de execução*)**

Algoritmos → Programas

Como transformar algoritmos em programas?

Linguagem de Programação

Uma linguagem de programação envolve dois aspectos principais:

- **Sintaxe: descreve a forma das expressões e comandos.**
  Ex: if (<*expressão*>) <*instrução*>

- **Semântica: descreve o significado das expressões e comandos.**
  Ex: se o valor atual da *expressão* for verdadeiro, a *instrução* será executada.

Tipos de Linguagem de Programação

Linguagem de máquina: conjunto de instruções que podem ser interpretadas e executadas diretamente pela CPU - utiliza códigos binários (0 e 1).

Linguagem de programação de baixo nível (*assembly*): de fácil tradução para a linguagem de máquina - baseada em mnemônicos, tais como: MOV (mover), ADD (adicionar, soma), MUL (multiplicar), JMP (pular, saltar).

Linguagem de programação de alto nível (*C*, *Java*, *Python*): precisa ser compilada ou interpretada para ser executada pela máquina – mais próxima da linguagem natural (*if*, *while*, *for*).

Programa “*Hello World*”

Linguagem de máquina

```
B4 40 BB 01 00 B9 0B 00 BA 14 01 CD 21 B0 01 B4
4C CD 21 00 68 65 6C 6C 6F 20 77 6F 72 6C 64
```

Linguagem de baixo nível (*Assembly*)

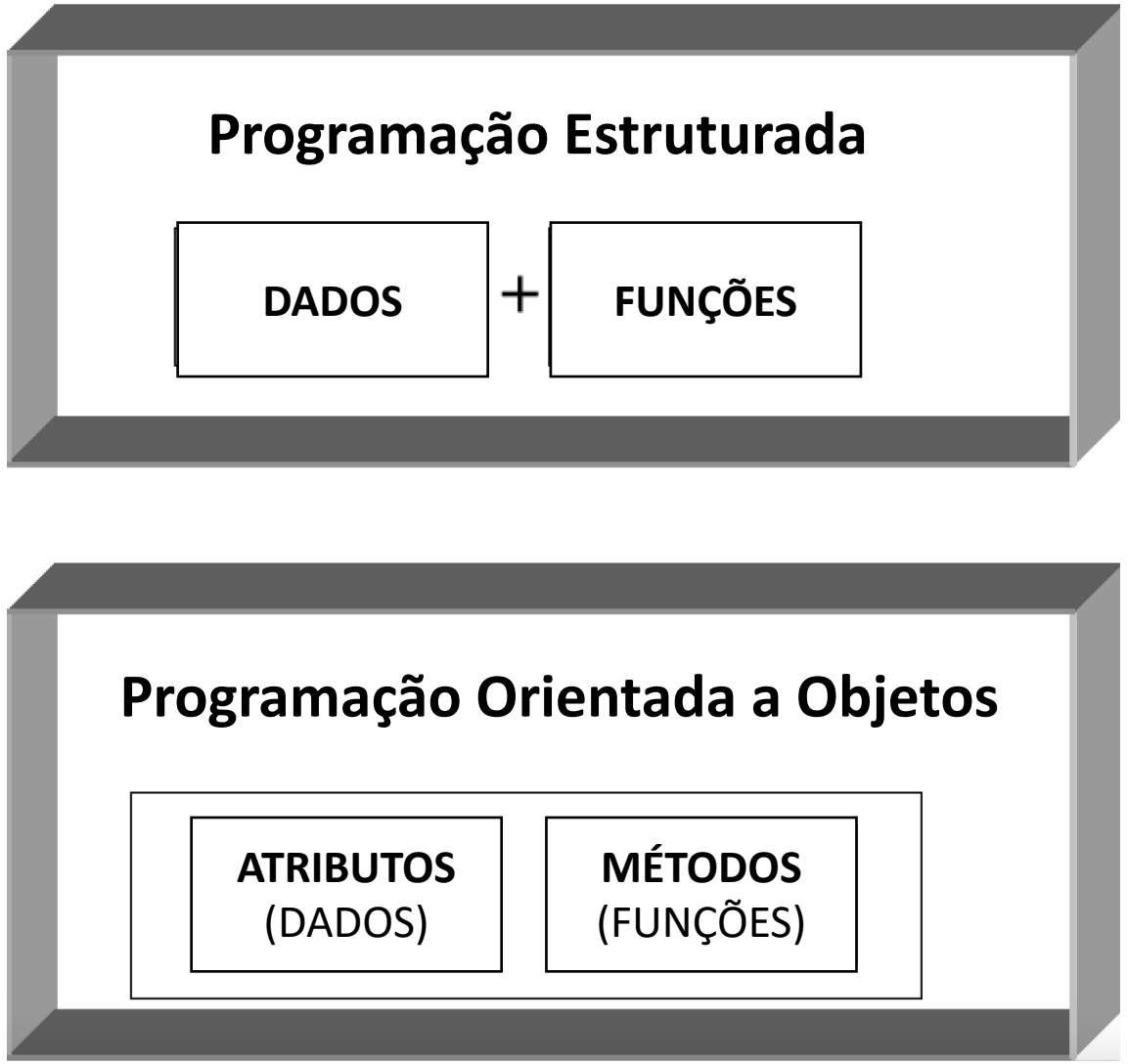
```
ORG 100h
section .text
        MOV AH, 40h
        MOV BX, 1
        MOV CX, 11
        MOV DX, msg
        INT 21h
        MOV AL, 1
        MOV AH, 4Ch
        INT 21h
section .data
        msg db "hello world"
```

Linguagem de alto nível (*C*)

```
1  #include <stdio.h>
2  int main() {
3  printf(“Hello, world!\n”);
4  return 0;
5  }
```

https://crg.eti.br/post/assembly-assembler-e-linguagem-de-maquina/

Paradigmas de Linguagem de Programação



C - C++ - C#

"C - Linguagem estruturada, considerada por muitos a melhor linguagem de programação existente. Ela foi usada para desenvolver os SO's mais conhecidos como: o Unix, Linux e Windows. Essa linguagem é robusta, pois permite trabalhar diretamente com os endereços de memória (ponteiros), o que possibilita aos programadores experientes o desenvolvimento de programas com um ótimo gerenciamento de memória.

C++ - Linguagem que permite programação estruturada e orientada a objetos. Ela é uma evolução do C. Tem todas as vantagens do C e possibilita a criação de objetos. Hoje o Windows e outros softwares da Microsoft são escritos em C++.

C# (lê-se "C-Sharp") - Linguagem Orientada a Objetos. C# é linguagem base do .NET, tecnologia da Microsoft que concorre com o Java."

Fonte: https://social.msdn.microsoft.com/Forums/vstudio/pt-BR/49daf64b-7070-4ed4-aa20-8989ab500743/diferena-entre-a-linguagem-c-e-a-c?forum=vsvbasicpt

Exemplo de código em C - C++ - C#

C

```c
#include <stdio.h>

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

C++

```cpp
#include <iostream.h>

int main()
{
cout << "Hello World" << endl;
return 0;
}
```

C#

```csharp
using System;

class HelloWorld
{
public static void Main(String args[])
{
Console.WriteLine("HelloWorld");
}
}
```

Compiladores e Interpretadores

Compiladores (Linguagens compiladas: C, C++, Delphi, etc)

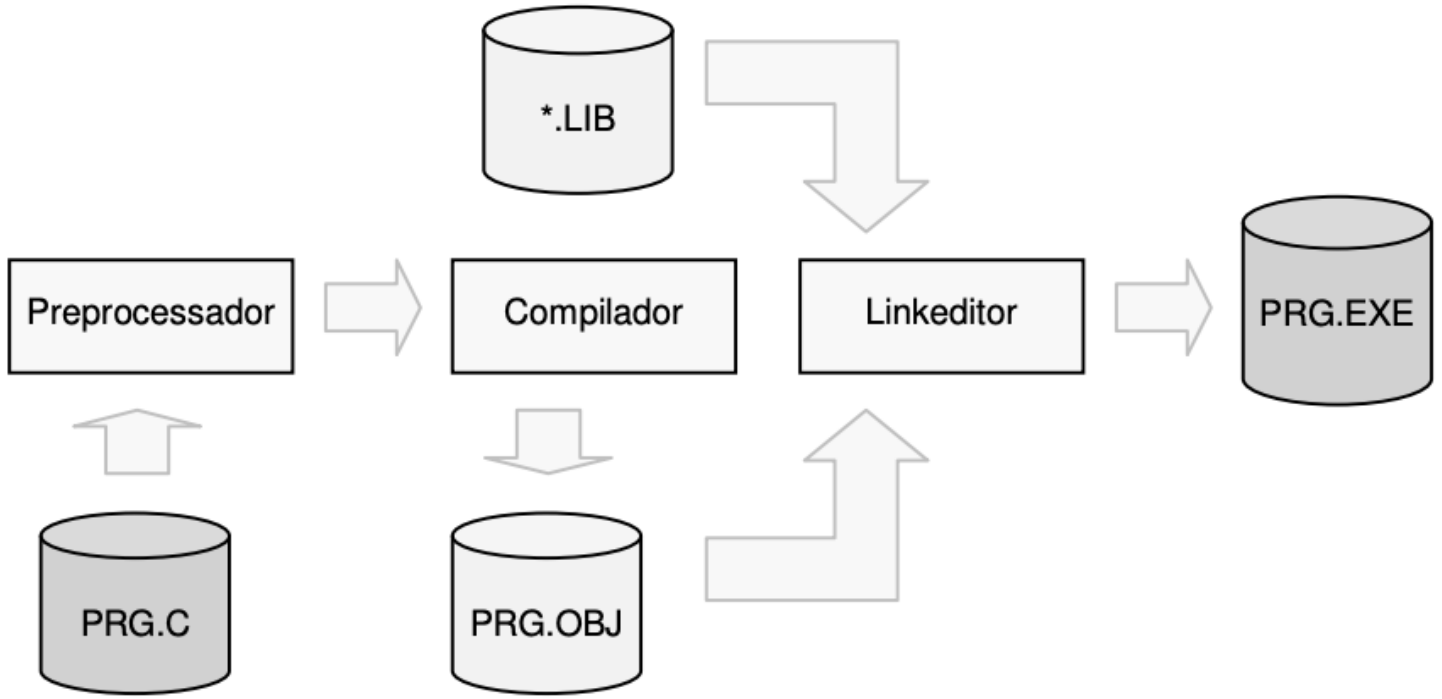
Lê a instrução-1 → confere a instrução → converte-a para linguagem de máquina.
Lê a instrução-2 → confere a instrução → converte-a para linguagem de máquina.
...
Lê a instrução-*n* → confere a instrução → converte-a para linguagem de máquina.

} **Arquivo Executável**

Interpretadores (Linguagens interpretadas: Java, C#, Python, etc.)

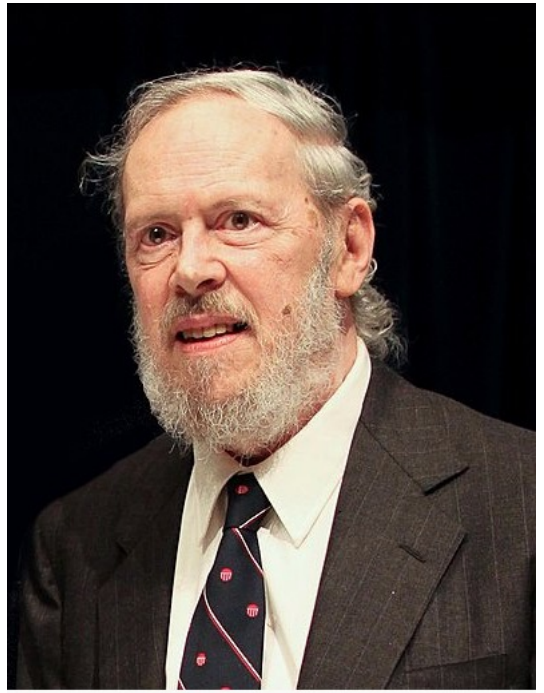
Lê a instrução-1 → confere a instrução → converte-a para linguagem de máquina → Executa a instrução.
Lê a instrução-2 → confere a instrução → converte-a para linguagem de máquina → Executa a instrução.
...
Lê a instrução-*n* → confere a instrução → converte-a para linguagem de máquina → Executa a instrução.

Processo de Geração de um Arquivo Executável



Linguagem de Programação C

Laboratórios da Bell Labs em Berkeley Heights, New Jersey - USA



Dennis Ritchie em 2011

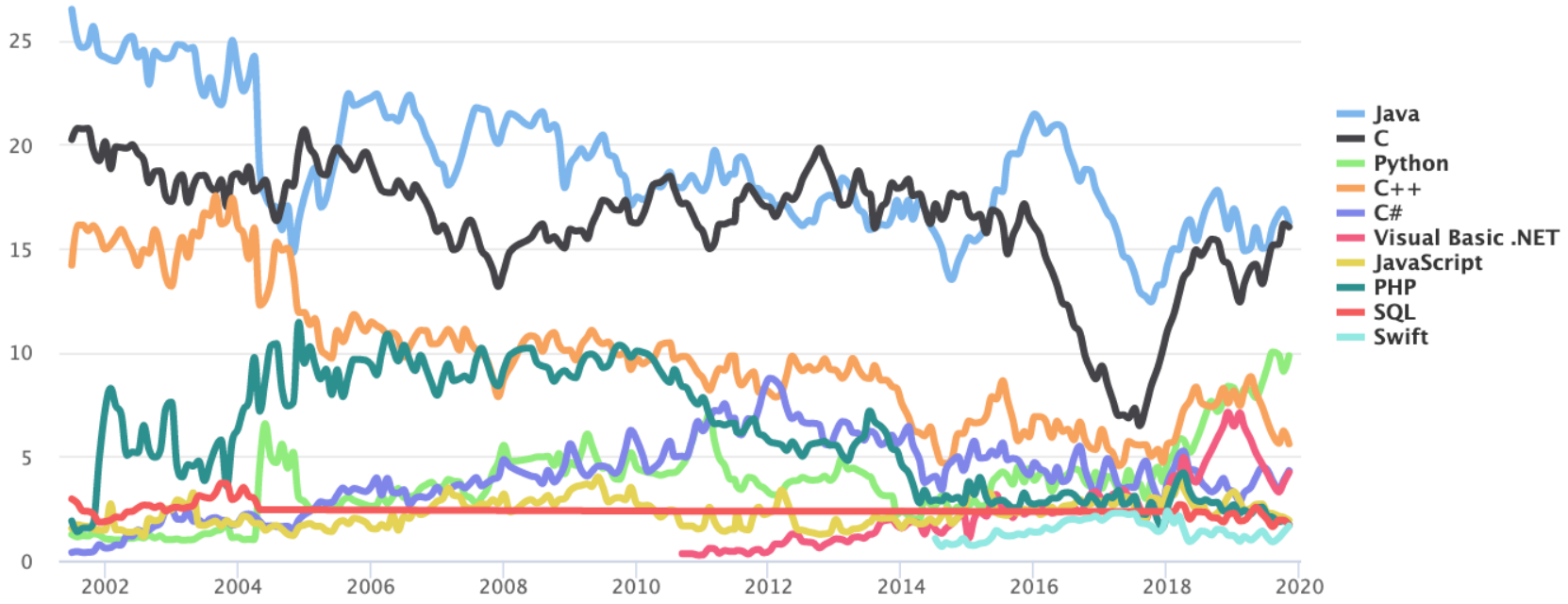
Data da criação = 1972 – Por quê "C"? – Concepção inicial

Fonte: https://pt.wikipedia.org/wiki/Bell_Labs

Fonte: https://pt.wikipedia.org/wiki/Dennis_Ritchie

Índice TIOBE para Novembro de 2019

Posição	Linguagem	Ratings [%]
1	Java	16,24
2	C	16,03
3	Python	9,84
4	C++	5,60
5	C#	4,31
6	Visual Basic. NET	4,22
7	JavaScript	1,92
8	PHP	1,70
9	SQL	1,69
10	Swift	1,65



A linguagem C é a segunda mais utilizada pelo mercado

Fonte: https://www.tiobe.com/tiobe-index//

Características da Linguagem C

- Linguagem de alto nível
- Popular
- Flexível em termos de aplicação
- Eficiente e robusta
- Controle total ao programador

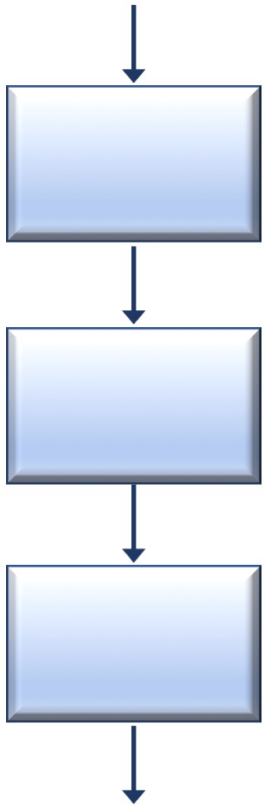
ANSI / ISO / IEC - C

- ANSI C ou C89
- ISO/IEC 9899:1990 – C90
- ISO 9899:1999 – C99
- ISO/IEC 9899:2011 – C11
- ISO/IEC 9899:2018 – C18

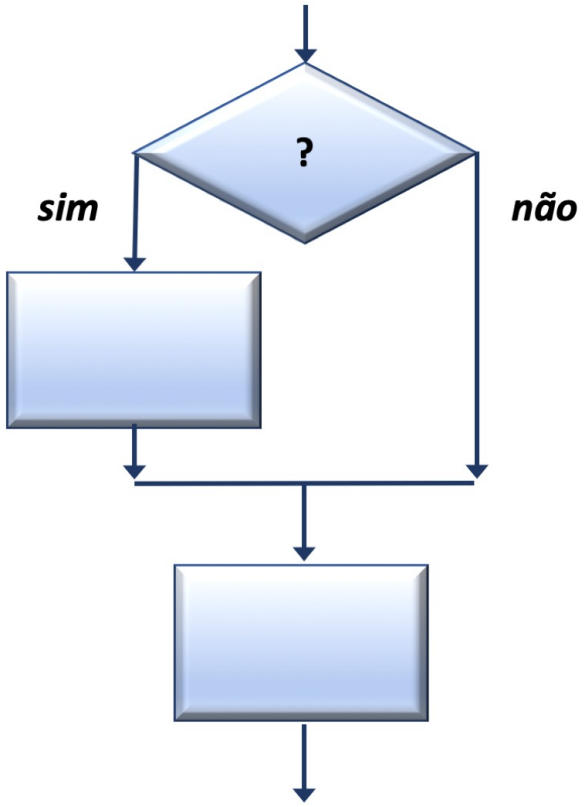
ANSI = American National Standards Institute
ISO = International Organization of Standardization
IEC = International Electrotechnical Commission

Fonte: https://pt.wikipedia.org/wiki/C_(linguagem_de_programa%C3%A7%C3%A3o)

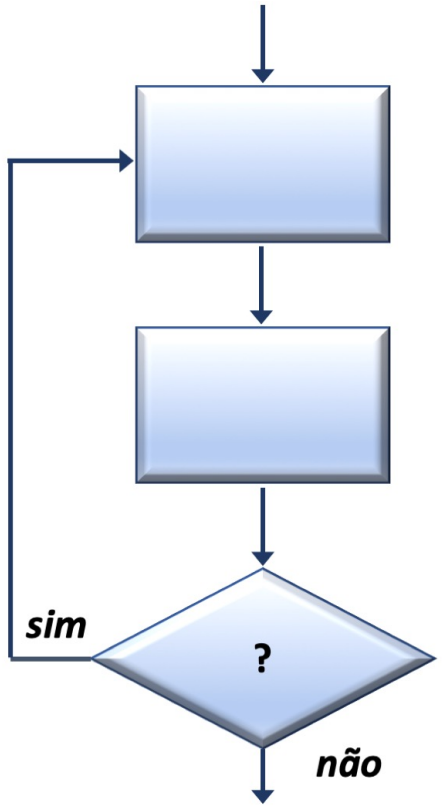
Linguagem de Programação C



Estrutura de Controle Sequencial



Estrutura de Controle Condicional



Estrutura de Controle Repetitiva

Introdução à Linguagem C

```
1.  // Meu primeiro programa em C
2.  #include <stdio.h>
3.  int main()
4.  {
5.      printf ("Hello World");
6.      return 0;
7.  }
```

- **Um programa em C é um arquivo texto, chamado código-fonte, que contém a tradução de um algoritmo para a linguagem C.**
- **Este arquivo contém as declarações e as instruções do programa, escritas conforme a sintaxe da linguagem.**
- **Todas as instruções devem estar dentro das "{}" que iniciam e encerram a função principal do programa: *main*(), são executadas na ordem em que são escritas e são encerradas com ";".**
- **Comentários de uma única linha iniciam-se com //. Comentários de múltiplas linhas devem estar entre /* e */.**

Primeiro Programa em C

Código-fonte

```
1.  // Meu primeiro programa em C
2.  #include <stdio.h>
3.  int main()
4.  {
5.      printf ("Hello World");
6.      return 0;
7.  }
```

Saída

```
Hello World
```

- **Linha 1 = Comentário → porção do texto desconsiderada pelo compilador.**
- **Linha 2 = Inclui a biblioteca padrão de entrada e saída <stdio.h>.**
- **Linhas 3 a 7 = Função principal do programa.**
- **Linha 5 = Função que imprime na tela o texto entre aspas.**

Executando o Programa em C

Supondo um ambiente Linux/Unix e o compilador gcc (*GNU C Compiler*)

Para compilar o programa:

$ gcc <*nome do arquivo código-fonte*> -o <*nome do arquivo executável*>

Para executar o programa:

$ ./<*nome do arquivo executável*>

Alguns compiladores online: Ideone.com, Codepad, Codechef, JSFiddle, Jdoodle, OnlineGDB, GCC Explorer.

IDEs - *Integrated Development Environment*

- **Editor de texto**
- **Compilador**
- **Linkeditor**
- **Depurador**

Exemplos de IDEs para C/C++:

Dev C++, Code: :Blocks, C++ Builder e Visual Studio C++.

Tipos de Erros em Programas

- **Erros de Compilação: referem-se a erros de digitação ou de sintaxe da linguagem. Neste caso, o arquivo executável não é gerado.**
  Ex: falta "}", ";"...

- **Erros de Linkedição: ocorrem quando há algum problema com o uso de bibliotecas necessárias ao programa principal. Neste caso, o arquivo executável também não é gerado.**

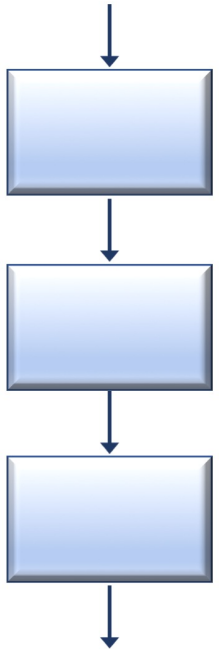
- **Erros de Execução (ou lógica): identificados quando o programa não apresenta o comportamento esperado. Neste caso, o arquivo executável foi gerado, mas provavelmente há um problema com o algoritmo.**
  Ex: a saída apresentada não está correta, a execução do programa é encerrada abruptamente, o programa entra em *loop* infinito, ...

Sumário

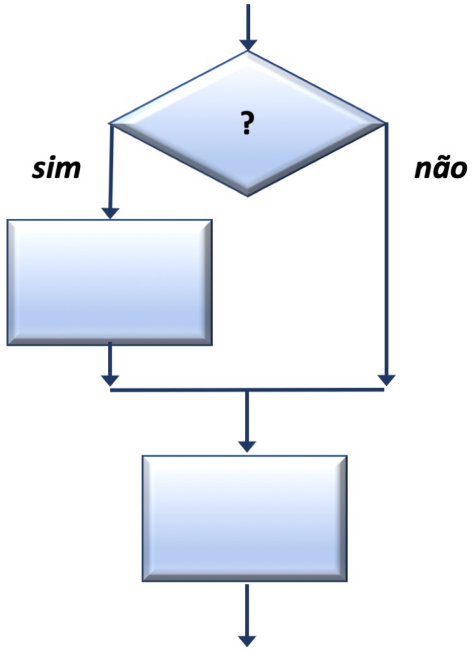
Tópicos preliminares

- **Tipos de dados (*char, int, float, double* e *void*)**
- **Identificadores**
- **Constantes (*diretiva define e operador const*)**
- **Simulando e verificando a memória com o operador *const***
- **Variáveis (*nomes, declaração e atribuição*)**
- **Simulando e verificando a memória com variáveis**
- **Ponto flutuante (*float* e *double*)**
- **Modificadores dos tipos de dados (*signed*, *unsigned*, *long*, *short*)**
- **Verificando o tamanho dos tipos de dados na memória**

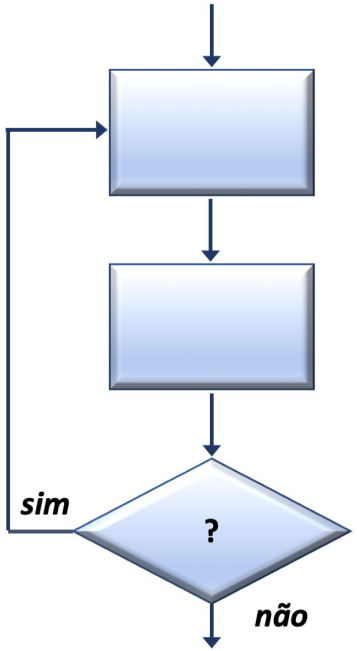
Programação Estruturada



Estrutura de Controle Sequencial



Estrutura de Controle Condicional



Estrutura de Controle Repetitiva

As instruções são executadas uma após a outra. Não existe desvio na sequência das instruções. Cada instrução é executada uma única vez. A ordem das instruções é importante.

Tipos de Dados

A linguagem C possui cinco tipos de dados primitivos (pré-definidos). O tipo de dado determina o espaço de memória que deve ser alocado (as especificações de cada tipo dependem do computador e do compilador utilizado).

Palavra Reservada	Tipo de Dado	Exemplo de espaço na memória
char	Caracter	1 *byte*
int	Número inteiro	2 *bytes*
float	Número real (precisão de até 6 dígitos)	4 *bytes*
double	Número real com precisão maior do que a de float (precisão de até 14 dígitos)	8 *bytes*
void	Vazio (sem valor)	1 *byte*

Identificadores
(dados = identificador + forma + tipo)

Representam os nomes escolhidos para rotular as constantes, as variáveis e as funções do programa.

Podem ser qualquer palavra que:

- **Não seja uma palavra reservada da linguagem (ex.: *if*, *printf*, *return*, ...).**
- **Seja formada apenas por letras, números e underline "_".**
- **Não inicie com número.**
- **Não contenha espaços entre os caracteres.**

Observações:

- **C é uma linguagem *case-sensitive*, ou seja, faz diferença entre nomes com letras maiúsculas e nomes com letras minúsculas ("Peso" e "peso" são identificadores diferentes).**
- **Deve-se usar nomes significativos dentro do contexto do programa.**
- **As constantes, variáveis e funções devem ser declaradas antes de serem utilizadas.**

Constantes

São identificadores associados a valor fixo e inalterável, ou seja, valor que não sofre modificação ao longo da execução do programa. Recomenda-se utilizar letras maiúsculas para declarar constantes.

Uma constante pode ser:

- **um número (inteiro ou real),**
- **uma sequência de caracteres (texto) ou**
- **um valor lógico (verdadeiro ou falso).**

Constantes são definidas com a diretiva *define* ou com o comando *const*.

Diretiva "*define*" e Comando "*const*"

Diretiva "*define*"

Sintaxe: #define <nome_da_constante> <valor>

- O pré-processador substitui todas as ocorrências do identificador ao longo do programa pelo seu valor antes da compilação;
- Não usa espaço da memória RAM;
- A declaração da constante deve ser feita no início do programa;
- Facilita a manutenção do código e torna o programa mais legível.

```
#include <stdio.h>
#define PI 3.14
int main()
{
   ...
   comp = 2 * PI * raio;
   return 0;
}
```

Comando "*const*"

Sintaxe: const <tipo_de_dado> < nome_da_constante > = <valor>;

- Declara uma constante para um tipo de dado específico;
- Usa espaço na memória RAM;
- O programa não compila se houver no código novo comando de atribuição para a constante (*assignment of read-only variable*).

```
#include <stdio.h>
int main()
{
const float PI = 3.14;
   ...
   comp = 2 * PI * raio;
   return 0;
}
```

Constantes com *printf* ()

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      printf ("A instrução A foi repetida 3 vezes. \n");
5.      printf ("A instrução %c %s %d vezes.\n",'A',"foi repetida",3);
6.      return 0;
7.  }
```

```
A instrução A foi repetida 3 vezes.
A instrução A foi repetida 3 vezes.
```

Operador *const* - Simulando e Verificando a Memória

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      const int x = 10;
5.      printf (" Endereço de x = %p \n", &x);
6.      printf (" Conteúdo de x = %d \n\n", x);
7.      return 0;
8.  }
```

Memória		
endereço	**const**	**conteúdo**
0x7ffcc25158ec	x	10

```
Endereço de x = 0x7ffcc25158ec
Conteúdo de x = 10
```

Variáveis

São identificadores cujo conteúdo pode variar durante a execução do programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um único valor a cada instante.

Uma variável pode ser:

- **um número (inteiro ou real),**
- **uma sequência de caracteres (texto) ou**
- **um valor lógico (verdadeiro ou falso).**

Variáveis devem ser declaradas e a elas deve ser atribuído um valor.

Declaração de variáveis

Declarar uma variável significa reservar um espaço de memória (endereço) para armazenar seu valor de acordo com o tipo de dado definido na instrução de declaração.

Sintaxe: <tipo-de-dado> <nome-da-variável>

Exemplos

- char letra;
- int num;
- float valor;
- double rendimento;
- int num1, num2;

Atribuição a variáveis

Atribuir um valor à uma variável significa fornecer a ela tal valor para que seja armazenado na memória. Uma atribuição só será válida se o valor atribuído for do mesmo tipo de dado da variável.

Sintaxe: < nome-da-variável > = <valor-da-variável>

Exemplos

- letra = ‘a’;
- num = 1;
- valor = 2.45;
- rend = 0.287651;
- num1 = 10, num2 = 20;

Observação: Uma única instrução pode conter tanto a declaração quanto a atribuição a variáveis.
Exemplo: int num = 1;

A importância do Nome das Variáveis

```
#include <stdio.h>

int main(){
     int abacaxi = 783;
     float acerola = 2500.00;
     float pitanga = 3750.00;
     float melancia = acerola + pitanga;

     printf("O valor calculado eh: %f\n", melancia);
     return0;
}
```

```
#include <stdio.h>

int main(){
     int codigoDoFuncionario = 783;
     float salarioBase = 2500.00;
     float comissoesRecebidas = 3750.00;
     float salarioDoMes = salarioBase +
comissoesRecebidas;

     printf("O valor calculado eh: %f\n", salarioDoMes);
     return0;
}
```

O nome de uma variável deve indicar claramente o seu propósito!

Variável com *printf* ()

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int x;
5.      x = 10;
6.      printf ("Valor de x = %d", x);
7.      return 0;
8.  }
```

```
Valor de x = 10
```

Variável – Simulando a Memória (1/2)

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int x;
5.      x = 10;
6.      return 0;
7.  }
```

Memória		
endereço	**variável**	**conteúdo**
0x7ffcd229bc5c	x	

Variável – Simulando a Memória (2/2)

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int x;
5.      x = 10;
6.      return 0;
7.  }
```

Memória		
endereço	**variável**	**conteúdo**
0x7ffcd229bc5c	x	10

Variável - Verificando a Memória

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int x;
5.      printf (" Endereço de x = %p \n", &x);
6.      printf (" Conteúdo de x = %d \n\n", x);

7.      x = 10;
8.      printf (" Endereço de x = %p \n", &x);
9.      printf (" Conteúdo de x = %d \n", x);

10.     return 0;
11. }
```

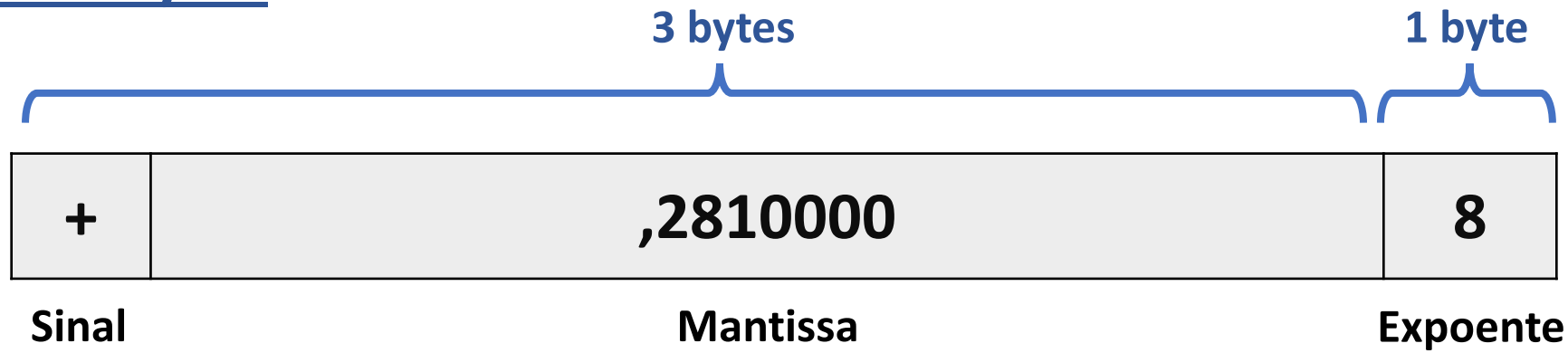
```
Endereço de x = 0x7ffcd229bc5c
Conteúdo de x = 0

Endereço de x = 0x7ffcd229bc5c
Conteúdo de x = 10
```

Ponto Flutuante – *Float* e *Double* (1/2)

Os tipos de dados *float* e *double* são usados para armazenar números reais. Eles são representados de forma similar à notação científica.

Ex.: + 28.100.000 = + 0,281 x 10^8

Supondo 4 bytes:



Ponto Flutuante – *Float* e *Double* (2/2)

```
#include <stdio.h>
int main()
{
   float a = 100.73;
   float b = 100.0;
   printf("%f",a - b);
   return 0;
}
```

```
0.730003
```

```
#include <stdio.h>
int main()
{
   float a = 1000.73;
   float b = 1000.0;
   printf("%f",a - b);
   return 0;
}
```

```
0.729980
```

```
#include <stdio.h>
int main()
{
   float a = 10000.73;
   float b = 10000.0;
   printf("%f",a - b);
   return 0;
}
```

```
0.730469
```

```
#include <stdio.h>
int main()
{
  float a = 10000000.73;
  float b = 10000000.0;
  printf("%f",a - b);
  return 0;
}
```

```
1.000000
```

O número de bits disponível para representar a mantissa é finito,
então a precisão que se consegue representar em computador é limitada.

Modificadores dos Tipos de Dados

Alteram o tamanho do tipo de dado e/ou sua forma de representação. A linguagem C disponibiliza quatro modificadores de tipo de dado: *signed*, *unsigned*, *long* e *short*.

Exemplos:

- **signed int (corresponde ao padrão de utilização do *bit* de sinal – raramente usado)**
- **unsigned int (o *bit* de sinal é usado para representar um valor)**
- **long int (número inteiro com domínio ampliado)**
- **short int (número inteiro com domínio reduzido)**
- **unsigned short int (número inteiro sem sinal e com domínio reduzido)**

Observações:

- **Os modificadores podem prefixar apenas os tipos *char*, *int* e *double*.**
- **Uma única declaração pode combinar até dois modificadores.**

Modificadores dos Tipos de Dados

O *bit* mais à esquerda em uma variável do tipo *int*, denominado *bit* de sinal, é normalmente utilizado pelo computador para distinguir entre valores positivos e negativos (0 = valor positivo e 1 = valor negativo). Usando o modificador *unsigned*, informamos ao compilador que somente valores sem sinal serão usados e que, portanto, não é necessário ter um *bit* de sinal. Com isso, podemos usar esse *bit* para representar valores e, consequentemente, a escala de valores dobra.

O modificador *signed* indica que os valores devem ser sinalizados; mas, como este é o caso normal, raramente ele é utilizado. O modificador *long* faz com que o espaço de memória reservado para uma variável do tipo *int* seja duplicado e, consequentemente, aumenta a capacidade de armazenamento da variável. Já o modificador *short*, em algumas máquinas, faz com que esse espaço reduza para a metade.

Verificando o Tamanho dos Tipos na Memória

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      printf (" char:   ");
5.      printf (" %ld ", sizeof(char));
6.      printf (" %ld ", sizeof(signed char));
7.      printf (" %ld \n", sizeof(unsigned char));
8.      printf (" int:    ");
9.      printf (" %ld ", sizeof(int));
10.     printf (" %ld ", sizeof(signed int));
11.     printf (" %ld ", sizeof(unsigned int));
12.     printf (" %ld ", sizeof(long int));
13.     printf (" %ld  \n", sizeof(short int));
14.     printf (" float:  ");
15.     printf (" %ld \n", sizeof(float));
16.     printf (" double: ");
17.     printf (" %ld ", sizeof(double));
18.     printf (" %ld \n", sizeof(long double));
19.     printf (" void:   ");
20.     printf (" %ld \n", sizeof(void));
21.     return 0;
22. }
```

```
char:     1   1   1
int:      4   4   4   8   2
float:    4
double:   8   16
void:     1
```

Atividade

1) **Implemente um código com as seguintes características:**
   - **Tenha como comentário a frase: Meu primeiro programa.**
   - **Defina uma constante chamada letra e atribua a ela o caractere A.**
   - **Mostre na tela o endereço e conteúdo da sua constante na memória.**
   - **Defina uma variável chamada num do tipo número inteiro.**
   - **Mostre na tela o endereço e conteúdo da sua variável na memória.**
   - **Atribua à sua variável o valor 10.**
   - **Mostre na tela o endereço e conteúdo da sua variável na memória após a atribuição do valor.**
   - **Por fim, apresente na tela o tamanho da sua constante e da sua variável na memória.**

OBS: não esqueça do include, main () e return.

Referências

- **FORBELLONE, A. L. V., EBERSPACHER, H. F. Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados, 3ª Edição, São Paulo, Pearson Prentice Hall, 2005.**
- **SOUZA, M. A. F., GOMES, M. M., SOARES, M. V., CONCILIO, R. Algoritmos e Lógica de Programação, 3ª Edição, São Paulo, Cengage, 2019.**
- **PUGA, S., RISSETTI, G. Lógica de Programação e Estrutura de Dados, 3ª. Edição, Prentice Hall, 2016.**
- **DEITEL, H. M., DEITEL, P. J. C: Como Programar. LTC, 2011.**
- **ASCENCIO, A. F. G., CAMPOS, E. A. V. Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++. Pearson Prentice Hall, 2012.**
- **VAREJÃO, F. M. Introdução à Programação: Uma nova abordagem usando C. Campus, 2015.**
- **CELES, W., CERQUEIRA, R., RANGEL, J. L. Introdução a Estrutura de dados com Técnicas de Programação em C. Campus, 2016.**
- **MIZRAHI, V. V. Treinamento em Linguagem C – Curso Completo, 2ª Edição, Pearson Makron Books, 2008.**
- **SCHILDT, H., C Completo e Total, 3ª Edição, Makron Books, 1997.**