# Linguagem C

## 1 - Histórico

A linguagem C foi criada por Dennis M. Ritchie e Ken Thompson nos laboratórios Bell em 1972, baseada na linguagem B de Thompson que era uma evolução da antiga linguagem BCPL. Embora tenha sido pensada com o propósito exclusivo de ser usada no desenvolvimento de uma nova versão do sistema operacional Unix, hoje é aplicada nos mais variados tipos de projetos.

Considerada uma linguagem de alto nível genérica, a linguagem C pode ser usada em diversos tipos de projeto, como a criação de aplicativos, sistemas operacionais, drivers, entre outros. Trata-se de uma linguagem estruturada que se tornou muito popular nos anos 80 - tanto que é difícil encontrar arquiteturas para as quais não existam compiladores para a linguagem C, o que garante o seu elevado nível de portabilidade.

Uma das grandes vantagens dessa linguagem é a capacidade de gerar códigos rápidos com um tempo de execução baixo. Além disso, a programação em C é bastante simplificada, pois sua estrutura é simples e flexível. Tendo isso em mente, podemos dizer que as principais características da linguagem C são a portabilidade, a geração de código eficiente, a simplicidade, a confiabilidade e facilidade de uso.

A linguagem C ainda é uma das mais populares do mercado devido às diversas vantagens que apresenta.

Alguns marcos históricos:

- 1969 – Desenvolvimento do UNIX (em um computador PDP 7 em linguagem Assembly).
- 1969 – Desenvolvimento da linguagem BCPL (muito próxima do Assembly).
- 1970 – Denis Ritchie cria a linguagem B a partir do BCPL nos laboratórios da Bell Telephones.
- 1971 – Primeiro desenvolvimento da linguagem C, sucessora do B (o C é a 2a letra de BCPL).
- 1973 – O sistema operacional UNIX é reescrito em linguagem C.
- 1978 – Primeira edição do livro “The C Programming Language”, Kernighan & Ritchie.
- 1980 – A linguagem C é padronizada pelo American National Standard Institute: surge o ANSI C.
- 1992 – Surge o C++, uma evolução da linguagem C incorporando conceitos da orientação a objetos.

## 2 - Compiladores e linguagem C

Um compilador é um programa de sistema que traduz um programa descrito em uma linguagem de alto nível para um programa equivalente em código de máquina para um processador. Em geral, um compilador não produz diretamente o código de máquina, mas sim um programa em linguagem simbólica semanticamente equivalente ao programa em linguagem de alto nível. O programa em linguagem simbólica é então traduzido para o programa em linguagem de máquina através de montadores.

Entre os exemplos de compiladores para a linguagem C temos o Dev C++ (https://sourceforge.net/projects/orwelldevcpp/) e Code::Blocks (https://www.codeblocks.org/). Existe também a opção de compiladores online como o Programiz (https://www.programiz.com/c-programming/online-compiler/).

## 3 - Estruturas básicas

```
/* Portugol */

ALGORITMO nome_do_algoritmo
VAR
  /* declaração de variáveis */
INICIO
  /* inicialização de variáveis */
  /* desenvolvimento (fórmulas, estruturas de decisão ou repetição, ...) */
FIM
```

```
/* Linguagem C */

#include <stdio.h>

int main(){

  /* declaração de variáveis */
  /* inicialização de variáveis */
  /* desenvolvimento (fórmulas, estruturas de decisão ou repetição, ...) */

  return 0;
}
```

## 4 - Declaração de variáveis

```
/* Portugol */

INTEIRO: a, b, c;
REAL: d, e;
CARACTER: f;
LÓGICO: g;
```

```
/* Linguagem C - A linguagem C não possui nativamente a representação para valores
lógicos (bool). Para usar uma váriável lógica, faz-se necessário a inclusão da
biblioteca <stdbool.h> */

int a, b, c;
float d, e;
char f;
bool g;
```

# 5 - Operadores

## 5.1 - Operador de atribuição

O operador de atribuição é o sinal "=". Seu uso em programação é ligeiramente diferente de seu uso na matemática normal. Se escrevermos

*x = y;*

em um programa em C, isto significa que o valor de y deve ser atribuído em x, e não que y é igual a x como seria de se esperar. Em uma instrução de atribuição, o lado direito pode ser qualquer expressão e o lado esquerdo deve necessariamente ser o nome de uma variável. A sintaxe, portanto, é:

*variavel = expressao;*

O operador de atribuição também pode ser usado de forma encadeada:

*bananas = tomates = laranjas = 50;*

Nesse exemplo, será atribuído o valor 50 às variáveis bananas, tomates e laranjas.

## 5.2 - Operadores aritméticos

| Operador | Símbolo | Ação | Exemplo |
|---|---|---|---|
| Adição | + | Soma seus dois operandos | x + y |
| Subtração | − | Subtrai o segundo operando do primeiro operando | x − y |
| Multiplicação | * | Multiplica seus dois operandos | x * y |
| Divisão | / | Divide o primeiro operando pelo segundo operando | x / y |
| Módulo | % | Fornece o resto da divisão do primeiro operando pelo segundo operando | x % y |

## 5.3 - Operadores relacionais

| Operador | Símbolo | Pergunta respondida | Exemplo |
|---|---|---|---|
| Igual | == | Operando 1 é igual ao operando 2? | x == y |
| Maior que | > | Operando 1 é maior que o operando 2? | x > y |
| Menor que | < | Operando 1 é menor que o operando 2? | x < y |
| Maior ou igual a | >= | Operando 1 é maior ou igual ao operando 2? | x >= y |
| Menor ou igual a | <= | Operando 1 é menor ou igual ao operando 2? | x <= y |
| Diferente | != | Operando 1 é diferente do operando 2? | x != y |

### 5.4 - Operadores lógicos

| Operador | Símbolo | Verdadeiro quando | Exemplo |
|---|---|---|---|
| E | && | Expressão 1 **E** expressão 2 são verdadeiras | exp1 && exp2 |
| OU | \|\| | Expressão 1 **OU** expressão 2 são verdadeiras | exp1 \|\| exp2 |
| Não | ! | A expressão é falsa | !exp1 |

## 6 - Funções de entrada e saída

```
/* Portugol */

ESCREVA("Informe número: ");
LEIA(num);

ESCREVA("O dobro de ", num, " é igual a ", num * 2);
```

```
/* Linguagem C - É utilizada as seguintes especificações de formato para entrada e saída: %d (números inteiros), %f (números decimais), %c (char) e \n para quebra de linha. */

printf("Informe número: ");
scanf("%d", &num);

printf("O dobro de %d é igual a %d", num, num * 2);
```

## 7 - Exemplos

Escreva um algoritmo que leia dois números inteiros, calcule e mostre a sua soma.

```
ALGORITMO soma_de_dois_numeros
VAR
  INTEIRO: num1, num2, soma;
INICIO
  ESCREVA("Informe o primeiro número: ");
  LEIA(num1);

  ESCREVA("Informe o segundo número: ");
  LEIA(num2);

  soma ← num1 + num2;

  ESCREVA("O resultado da soma é: ", soma);
FIM
```

Escreva um programa em linguagem C que leia dois números inteiros, calcule e mostre a sua soma.

```c
#include <stdio.h>

int main(){

  int num1, num2, soma;

  printf("Informe o primeiro número: ");
  scanf("%d", &num1);

  printf("Informe o segundo número: ");
  scanf("%d", &num2);

  soma = num1 + num2;

  printf("O resultado da soma é: %d", soma);

  return 0;
}
```

# 8 - Estruturas condicionais

## 8.1 - Seleção simples

```
/* Portugol */

SE(condição) ENTÃO
  bloco_de_comandos_1;
SENÃO
  bloco_de_comandos_2;
FIM_SE
```

```c
/* Linguagem C */

if(condição){
  bloco_de_comandos_1;
}
else{
  bloco_de_comandos_2;
}
```

### 8.2 - Seleção composta

```
/* Portugol */

ESCOLHA (variável)
  CASO valor_1: bloco_de_comandos_1;
  CASO valor_2: bloco_de_comandos_2;
  CASO valor_3: bloco_de_comandos_3;
  ...
  CASO valor_n: bloco_de_comandos_n;
  CASO CONTRARIO: bloco_de_comandos_específico;
FIM_ESCOLHA
```

```
/* Linguagem C */

switch (variável){
  case valor_1: bloco_de_comandos_1;
  break;
  case valor_2: bloco_de_comandos_2;
  break;
  case valor_3: bloco_de_comandos_3;
  break;
  ...
  case valor_n: bloco_de_comandos_n;
  break;

  default: bloco_de_comandos_específico;
}
```

## 9 - Estruturas de repetição

### 9.1 - Repetição com teste no início

```
/* Portugol */

ENQUANTO(condição) FAÇA
  bloco_de_comandos;
FIM_ENQUANTO
```

```
/* Linguagem C */

while(condição){
  bloco_de_comandos;
}
```

### 9.2 - Repetição com teste no fim

```
/* Portugol */

REPITA
  bloco_de_comandos;
ENQUANTO(condição);
```

```
/* Linguagem C */

do{
  bloco_de_comandos;
}while (condição);
```

### 9.3 - Repetição com variável de controle

```
/* Portugol */

PARA(inicialização; condição; atualização) FAÇA
  bloco_de_comandos;
FIM_PARA
```

```
/* Linguagem C */

for(inicialização; condição; atualização){
  bloco_de_comandos;
}
```

## 10 - Vetores

```
/* Portugol */

/* Declaração de um vetor com capacidade para 5 números inteiros */
INTEIRO: numeros[5];

/* Inserção do valor 15 na primeira posição do vetor */
numeros[0] ← 15;

/* Recuperação de elemento do vetor e atribuição a uma variável */
x ← numeros[3];
```

```
/* Linguagem C */

/* Declaração de um vetor com capacidade para 5 números inteiros */
int numeros[5];
```

```
/* Inserção do valor 15 na primeira posição do vetor */
numeros[0] = 15;

/* Inserção de valores no vetor durante a sua declaração */
int numeros[5] = {15, 20, 25, 30, 35};

/* Inserção de valores no vetor durante a sua declaração, sem definição de tamanho do
vetor */
int numeros[] = {15, 20, 25, 30, 35};

/* Recuperação de elemento do vetor e atribuição a uma variável */
x = numeros[3];
```

## 11 - Matrizes

```
/* Portugol */

/* Declaração de uma matriz com 2 linhas e 3 colunas */
INTEIRO: numeros[2][3];

/* Inserção do valor 15 na primeira posição da matriz */
numeros[0][0] ← 15;

/* Recuperação de elemento da primeira posição da matriz e atribuição a uma variável */
x ← numeros[0][0];
```

```
/* Linguagem C */

/* Declaração de uma matriz com 2 linhas e 3 colunas */
int numeros[2][3];

/* Inserção do valor 15 na primeira posição da matriz */
numeros[0][0] = 15;

/* Inserção de valores na matriz durante a sua declaração */
int numeros[2][3] = {{15, 20, 25}, {30, 35, 40}};

/* Recuperação de elemento da primeira posição da matriz e atribuição a uma variável */
x = numeros[0][0];
```