Linguagem C
Projetada por Dennis M. Ritchie, em 1972, no laboratório da Bell.
Em 1973, Dennis M. Ritchie e Ken Thompson, escreveram uma versão do sistema operacional Linux utilizando a linguagem C.

main()
Principal função. Marca o ponto de inicialização do processo de execução do programa.

stdio.h
Biblioteca padrão. Inclui as funções printf, scanf, getchar, puts, gets, entre outras.

função printf()
Utilizada para saída de dados (no vídeo).
Códigos de formatação
%d Permite a escrita de números inteiros (base 10)
%f Permite a escrita de números reais (ponto flutuante)
%c Permite a escrita de apenas um caractere
%s Permite a escrita de uma série de caracteres (string)

```
#include "stdio.h"
main()
{
	int a = 10;
	int b = 20;
	int x = a + b;
	printf("%d", x);
}
```

função scanf()
Utilizada para entrada de dados (via teclado).

```
#include "stdio.h"
main()
{
	int a;
	int b;
	int x;
	scanf("%d", &a);
	scanf("%d", &b);
	x = a + b;
	printf("%d", x);
}
```

```
#include "stdio.h"
main()
{
     int a;
     int b;
     int x;
     printf("Informe o primeiro valor ... ");
     scanf("%d", &a);
     printf("Informe o segundo valor .... ");
     scanf("%d", &b);
     x = a + b;
     printf("%d", x);
}
```

```
#include "stdio.h"
main()
{
     int a;
     int b;
     int x;
     printf("Soma dois numeros inteiros \n");
     printf("Informe o primeiro valor ... ");
     scanf("%d", &a);
     printf("Informe o segundo valor .... ");
     scanf("%d", &b);
     x = a + b;
     printf("%d", x);
}
```

Exercício

```
#include "stdio.h"
/*
ht = horas trabalhadas
vh = valor por hora
pd = percentual de desconto
---
sb = salario bruto
vd = valor do desconto
sl = salario liquido

*/
main()
{
     float ht, vh, pd, sb, vd, sl;

     printf("Horas trabalhadas: ......... ");
     scanf("%f", &ht);
     printf("Valor por hora: ............ ");
     scanf("%f", &vh);
     printf("Percentual de desconto: .... ");
     scanf("%f", &pd);

     sb = ht * vh;
     vd = (pd/100) * sb;
     sl = sb - vd;

     printf("Salario Bruto: ......... %7.2f\n", sb);
     printf("Valor Desconto: ........ %7.2f\n", vd);
     printf("Salario Liquido: ....... %7.2f\n", sl);

}
```

Como resolver problemas com acentuação em português

```
#include <stdio.h>
#include <locale.h>

int main(void)
{
    setlocale(LC_ALL, "Portuguese");
    printf("Alô mundo! \n\n");
}
```

Estruturas de decisão

Operadores relacionais

```
==   !=   <   >   <=   >=
```

Operadores lógicos

```
&&   ||   !
```

```
#include "stdio.h"
#include <locale.h>
main()
{
      setlocale(LC_ALL, "Portuguese");
      int a;
      int b;
      printf("Verifica qual número inteiro é maior \n");
      printf("Informe o valor para a: ");
      scanf("%d", &a);
      printf("Informe o valor para b: ");
      scanf("%d", &b);
      if (a > b)
            printf("a é maior que b");
}
```

```
#include "stdio.h"
#include <locale.h>
main()
{
      setlocale(LC_ALL, "Portuguese");
      int a;
      int b;
      printf("Verifica qual número inteiro é maior \n");
      printf("Informe o valor para a: ");
      scanf("%d", &a);
      printf("Informe o valor para b: ");
      scanf("%d", &b);
      if (a > b)
            printf("a é maior que b");
      else if (b > a)
            printf("b é maior que a");
}
```

```
#include "stdio.h"
#include <locale.h>
main()
{
     setlocale(LC_ALL, "Portuguese");
     int a;
     int b;
     printf("Verifica qual número inteiro é maior \n");
     printf("Informe o valor para a: ");
     scanf("%d", &a);
     printf("Informe o valor para b: ");
     scanf("%d", &b);
     if (a > b)
          printf("a é maior que b");
     else if (b > a)
          printf("b é maior que a");
     else
          printf("a e b são iguais");

}
```

```
#include<stdio.h>
//Imprime x vale 10 e vale 20
void main()
{
int x=10, y=20;

    if(x == 10)
    {
     printf("x vale 10 ");
     if(y == 20)
        {
            printf("y vale 20");
        }
    }
}
```

```
#include<stdio.h>
//Imprime x vale 10
void main()
{
int x=10, y=30;

    if(x == 10)
    {
     printf("x vale 10 ");
     if(y == 20)
        {
            printf("y vale 20");
        }
    }
}


#include<stdio.h>
//Não imprime nada, pois a primeira condição é falsa
void main()
{
int x=30, y=20;

    if(x == 10)
    {
     printf("x vale 10 ");
     if(y == 20)
        {
            printf("y vale 20");
        }
    }
}


#include<stdio.h>
//Imprime x vale 10 e vale 20
void main()
{
int x=10, y=20;

    if(x==10 && y==20)
    {
     printf("Verdadeiro");
    }
}


** Demonstrar com ||
```

```
#include "stdio.h"
#include <locale.h>
main()
{
     setlocale(LC_ALL, "Portuguese");
     float a;
     float b;
    float media;
     printf("Calcula a média com base em duas notas \n");
     printf("Informe a 1a nota: ");
     scanf("%f", &a);
     printf("Informe a 2a nota: ");
     scanf("%f", &b);
     media = (a + b) / 2;
    printf("Média: %4.2f", media);
     if (media >= 6)
          printf("\nAPROVADO");
     else
          printf("\nREPROVADO");
}
```

Estruturas de repetição

while

Efetua um teste lógico no início de um looping.
Executa um conjunto de instruções enquanto a condição verificada for verdadeira.

```
/* while */
#include "stdio.h"
main()
{
     int i;
     i = 1;
     while (i <= 3)
     {
          printf("\ni =  %d", i);
          i++;
     }
}
```

```
/* while */
#include "stdio.h"
main()
{
     int a, b, s, i;
     i = 1;
     while (i <= 3)
     {
          printf("\n\nInforme um valor (inteiro) para a: ");
          scanf("%d", &a);
          printf("Informe um valor (inteiro) para b: ");
          scanf("%d", &b);
          s = a + b;
          printf("\ni =  %d", i);
          printf("\nResultado: %d", s);
          i++;
     }
}
```

do ... while

Efetua um teste lógico no final de um looping.
Executa um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida.

```
/* do ... while */
#include "stdio.h"
main()
{
      int i;
      i = 1;
      do
      {
            printf("\ni =  %d", i);
            i++;
      }
      while (i <= 3);
}
```

```
/* do ... while */
#include "stdio.h"
main()
{
      int a, b, s, i;
      i = 1;
      do
      {
            printf("\n\nInforme um valor (inteiro) para a: ");
            scanf("%d", &a);
            printf("Informe um valor (inteiro) para b: ");
            scanf("%d", &b);
            s = a + b;
            printf("\ni =  %d", i);
            printf("\nResultado: %d", s);
            i++;
      }
      while (i <= 3);
}
```

for

Executa a inicialização e testa a condição.

```
/* for */
#include "stdio.h"
main()
{
     int i;
     for (i=1;i<=3;i++)
     printf("i = %d\n", i);
}
```

```
/* for */
#include "stdio.h"
main()
{
     int a, b, s, i;
     for (i=1;i<=3;i++)
     {
          printf("\n\nInforme um valor (inteiro) para a: ");
          scanf("%d", &a);
          printf("Informe um valor (inteiro) para b: ");
          scanf("%d", &b);
          s = a + b;
          printf("\ni =  %d", i);
          printf("\nResultado: %d", s);
     }
}
```

```
/* for */
#include "stdio.h"
main()
{

     int i, j;
     printf("Informe a quantidade de repeticoes: ");
     scanf("%d", &j);
    for (i=1; i<=j; i++)
        printf("i = %d\n", i);
}
```

Matrizes

Matrizes são coleções indexadas de variáveis de mesmo tipo.

Matrizes de uma dimensão (Vetores)

Matrizes unidimensionais ou vetores apresentam apenas uma linha e várias "colunas".

```
// Exemplo com variáveis escalares:
#include "stdio.h"
main()
{
	float nota1, nota2, nota3, nota4;
	float soma = 0, media;
	printf("\nCalculo de media\n");
	printf("\nInforme as 4 notas\n");
	scanf("%f %f %f %f", &nota1, &nota2, &nota3, &nota4);
	soma = nota1 + nota2 + nota3 + nota4;
	media = soma / 4;
	printf("A media e igual a: %4.2f", media);
}
```

```
// Exemplo com vetor
#include "stdio.h"
main()
{
	float notas[4]; // vetor com quatro elementos do tipo float
	// 1° elemento notas[0], 2° elemento notas[1], ...
	float soma = 0, media;
	int i;
	printf("\nCalculo de media\n\n");
	for (i = 0; i <= 3; i++) {
		printf("Informe a %da. nota: ", i+1);
		scanf("%f", &notas[i]);
		soma += notas[i];
	}
	media = soma / 4;
	printf("A media e igual a: %4.2f", media);
}
```

Matrizes de duas ou mais dimensões

Matrizes unidimensionais ou vetores apresentam apenas uma linha e várias "colunas".

```
// Exemplo com matrizes multidimensionais
#include "stdio.h"
main()
{
      float notas[8][4];
      float numero;
      int i, j;
      printf("\nCalculo de media\n\n");
      for (i = 0; i <= 7; i++) {
            printf("Informe a nota do %do. aluno: ", i+1);
            for (j = 0; j <= 3; j++) {
                  printf("Nota %d: ", j+1);
                  scanf("%f", &numero);
                  notas[i][j] = numero;
            }
      }

      // Saida de notas

      for (i = 0; i <=7; i++) {
            printf("\nAs notas do aluno %d sao: \n\n", i+1);
            for (j = 0; j <= 3; j++) {
                  printf("Nota %d: %5.2f\n", j+1, notas[i][j]);
            }
      }
}
```

Strings

String, na linguagem C, é uma matriz unidimensional do tipo **char**.
Visto que String é uma matriz, então é possível acessar qualquer um dos seus elementos através dos seus respectivos índices.

```
#include "stdio.h"
main() {
      char nome[10];
      printf("Digite o seu nome: ");
      scanf("%s", &nome);  // ou: scanf("%s", nome);
      printf("Você digitou: %s", nome);
}
```

```
#include "stdio.h"
main() {
      char nome[10];
      puts("Digite o seu nome: ");
      gets(nome);
      printf("Você digitou: %s", nome);
}
```

```
//percorrendo um vetor de char
#include <stdio.h>
main() {
      int i;
      char texto[] = "string";

      printf("Valor da variavel texto = %s\n", texto);

      for (i=0; i<6; i++) {
            printf("Valor do elemento %d da string = %c\n",i, texto[i]);
      }

}
```

Estruturas (Matrizes Heterogêneas)

Matrizes trabalham com um único tipo de dado.
Estruturas em C (também chamadas de registro, em outras linguagens) podem trabalhar com tipos de dados diferentes. Cada elemento de um registro costuma ser denominado de campos.

```
#include "stdio.h"
main() {
      struct cad_aluno {
            char nome[30];
            float a1;
            float a2;
            float a3;
            float a4;
      };

      struct cad_aluno aluno;

      printf("Cadastro de Aluno\n\n");
      printf("Informe o nome ......: "); scanf("%s", aluno.nome);
      printf("Informe a 1a. nota ......: "); scanf("%f", &aluno.a1);
      printf("Informe a 2a. nota ......: "); scanf("%f", &aluno.a2);
      printf("Informe a 3a. nota ......: "); scanf("%f", &aluno.a3);
      printf("Informe a 4a. nota ......: "); scanf("%f", &aluno.a4);
      printf("\n");
      printf("Nome ......: %s\n", aluno.nome);
      printf("Nota 1 ....: %4.2f\n", aluno.a1);
      printf("Nota 2 ....: %4.2f\n", aluno.a2);
      printf("Nota 3 ....: %4.2f\n", aluno.a3);
      printf("Nota 4 ....: %4.2f\n", aluno.a4);

}
```

Funções

Uma função pode ser vista como **um conjunto de comandos que realiza uma tarefa específica**. Pode-se dizer que é um pequeno "programa" utilizado por outros programas.

A função é referenciada (chamada) pelo programa principal através de um **nome** atribuído a ela.

A utilização de funções, muito comum na programação estruturada, visa **subdividir um programa em partes (módulos) menores que realizam uma tarefa bem definida**.

Benefícios da utilização de funções:

- Permite o reaproveitamento de código já construído (por você ou por outros programadores);
- Evita que um mesmo trecho de código seja repetido várias vezes dentro de um mesmo programa e, com isso, qualquer alteração é feita apenas nesse trecho e de forma simples.
- Evita que os blocos do programa fiquem grandes demais e, por consequência, mais difíceis de entender;
- Facilita a leitura do programa de maneira que os blocos de código possam ser logicamente compreendidos de forma isolada.

Esqueleto de uma função

```
tipo_de_retorno nome_da_função (lista de parâmetros)
{
   instruções;
   retorno_da_função;
}
```

Tipo de retorno é o tipo de valor que a função retornará, por exemplo:

```
int nomefuncao(lista de parâmetros)
// a função retornará para o programa um valor do tipo int.
```

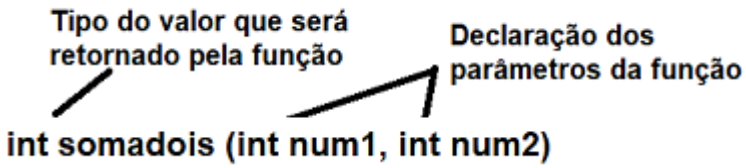
Parâmetros

A Lista de Parâmetros, também é chamada de Lista de Argumentos, é opcional.

Funciona como a interface de comunicação (passagem de valores/dados) entre o programa (chamador) e a função.

Os parâmetros de uma função são definidos como se estivesse declarando uma variável, entre os parênteses do cabeçalho da função.

Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas.



Existem duas maneiras de passar valor através dos parâmetros: por **valor** ou por **referência**.

Nesse momento, será apresentado a passagem de parâmetro por valor. Quando introduzirmos o conceito de Ponteiro, apresentaremos o outro tipo.

No exemplo a seguir temos a função SOMA que possui dois parâmetros, sendo o primeiro um float (a) e o segundo um int (b).

```
float soma(float a, int b)  // parâmetros separados por vírgulas
{
   float result;            // declaração de variáveis locais

   result = a + b;
   return result; // retorna para o programa o resultado da soma de a + b
}
```

Os parâmetros são passados para uma função de acordo com a sua posição. Ou seja, o primeiro parâmetro da chamada (programa) define o valor do primeiro parâmetro na definição da função, o segundo parâmetro do programa define o valor do segundo parâmetro da função e assim por diante.

Os nomes dos parâmetros na chamada não têm relação com os nomes dos parâmetros na definição da função. No código a seguir, por exemplo, a função **soma** é chamada recebendo como parâmetros as variáveis "a" e "b", nesta ordem.

```
#include <stdio.h>

float soma(float a, int b)
{
   float result;
   result = a + b;
   return result;
}

int main()
{
    float a;
    int b;
    float s;
    a = 10.3;
    b = 12;
    s = soma(a,b);  // Chamada da função soma(12.3,10);
    printf("A soma de %f com %d é %f\n", a,b,s);
    return 0;
}
```

Faça um programa para calcular a área de um retângulo utilizando uma função que receba como parâmetros a largura e o comprimento dessa figura geométrica. O programa deverá solicitar ao usuário informar essas duas medidas.

```
#include <stdio.h>

float CalculaArea(float largura, float comprimento){
      return largura * comprimento;
}

int main() {
      float larg, comp, area;

      printf("\nDigite a largura: ");
      scanf("%f", &larg);
      printf("\nDigite o comprimento: ");
      scanf("%f", &comp);

      area = CalculaArea(larg, comp);

      printf("Area = %f", area);

}
```

Faça um programa que leia duas notas de cada aluno numa turma de 5 alunos. Para cada aluno, calcular a média ponderadas das notas, sabendo que a nota1 tem peso = 4 e a nota2 tem peso = 6. Imprimir a média do aluno e o conceito final, conforme tabela abaixo:

INTERVALO	CONCEITO
0.0 a 4.9	D
5.0 a 6.9	C
7.0 a 8.9	B
9.0 a 10.0	A

```
#include <stdio.h>

float mediapond(float nota1, float nota2)
{
    float media;
    media = nota1 * 0.4 + nota2 * 0.6;
    return media;
}
```

```
char conceito(float media)
{
    char conc;
    if (media <= 4.9){
      conc = 'D';
    }
    else if (media <= 6.9) {
        conc = 'C';
    }
    else if (media <= 8.9 ){
            conc ='B';
    }
    else {
            conc = 'A';
    }

    return conc;
}

int main() {
  int i;
  float nota1, nota2, med;
  char conce;

  for (i=0; i < 5; i++) {
    printf("\n\nDigite a nota 1:");
    scanf("%f", &nota1);
    printf("\nDigite a nota 2:");
    scanf("%f", &nota2);
    med = mediapond(nota1,nota2);
    conce = conceito(med);
    printf("\n\nmédia = %f \nConceito final = %c", med, conce);
  }
}
```

Existem **dois métodos de passagem de parâmetros** para funções:

Passagem por valor: permite usar dentro de uma função uma cópia do valor de uma variável, porém não permite alterar o valor da variável original (somente a cópia pode ser alterada).

Passagem por referência: É passada para a função uma referência da variável, sendo possível alterar o conteúdo da variável original usando-se esta referência.

Na linguagem C a passagem por referência é implementada com o uso de ponteiros.

O próximo tópico abordará o que são e como utilizar ponteiros, inclusive para criar funções com passagem de parâmetro por referência.

Ponteiros

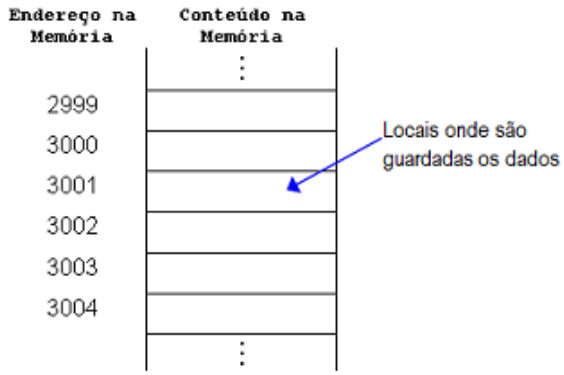
A linguagem C permite que em uma variável seja armazenado o endereço de outra variável. Para que isso seja possível é necessário trabalhar com uma variável como sendo um ponteiro, um apontador.

A declaração de um ponteiro ocorre de forma semelhante à de uma variável simples, tendo como diferencial o asterisco antes no nome da variável.

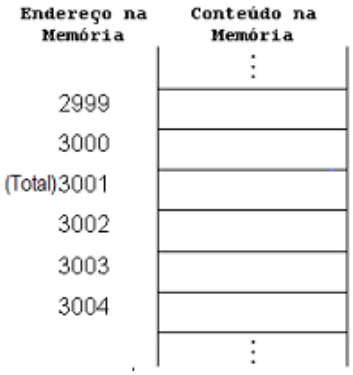
A memória é formada por um conjunto de "espaços" onde os dados são armazenados.

Esses "espaços" são identificados por nomes (interno nos programas), e por números ou "endereços" (externos aos programas), utilizados pelos programas para acessar os dados.

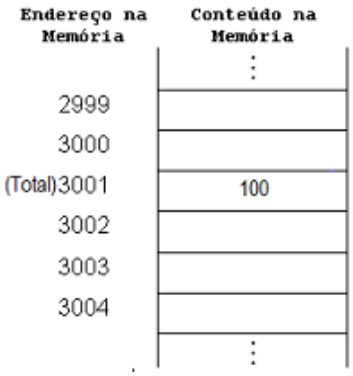
A figura abaixo exibe um trecho da memória:



Suponha que você declare uma variável denominada "Total" em seu programa. Essa variável será associada a um (ou mais) endereço(s) na memória (endereço 3001), conforme a figura abaixo:



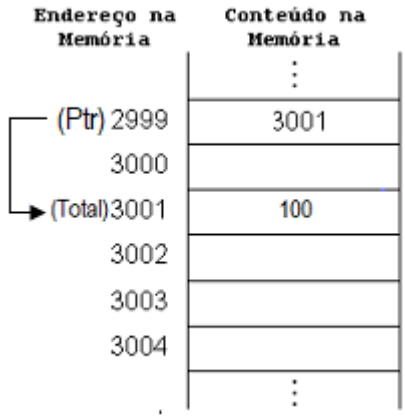
Quando você atribui o valor **100** para a variável **Total** o programa procura o endereço da variável e armazena o valor.



Agora podemos começar a definir Ponteiros.

Ponteiros são variáveis que armazenam endereços (de outras variáveis) ao invés de armazenarem valores. O conteúdo de uma variável ponteiro é o endereço de outra variável.

Suponha que você defina a variável **Ptr** como sendo ponteiro e armazene o endereço **3001**. Essa variável armazenará o endereço da variável **Total**, conforme figura abaixo:



Assim dizemos que a variável **Ptr** está "apontando" para variável **Total**, isto é, qualquer alteração em **Ptr** irá alterar o conteúdo de **Total**.

Os ponteiros são usados em muitas linguagens de programação para manipular cadeias de caracteres, passar parâmetros para funções, manipulação matrizes de dados e criação de listas ligadas e outras estruturas de dados complexas.

Ponteiros proporcionam uma grande flexibilidade para o gerenciamento de memória e otimização de programas.

```
#include "stdio.h"
int main(void)
{
	int x = 100;
	int *ptrx = &x; // *ptrx armazena o endereço de memória da variável x
	int y = 200;
	*ptrx = y; // *ptrx envia para o endereço da variável x o valor de y
	printf("%d\n",x);
	printf("%d\n",y);
}
```

Usando ponteiros é possível alterar os valores das variáveis passadas como argumentos para uma função.

```
#include <stdio.h>

void soma10(int a)
{
      a = a + 10;
      printf("Valor de a apos a soma = %d \n",a);
      return;
}

void soma10p(int *a)
{
      *a = *a + 10;
      printf("Valor de a apos a soma = %d \n",*a);
      return;
}

int main(void)
{
      int x = 20;

      printf("x vale: %d \n",x); // 20

      soma10(x);  //chamada da função com valor como parâmetro
      printf("Agora x vale: %d \n",x); // 20

      soma10p(&x);  //chamada da função com ponteiro como parâmetro
      printf("Agora x vale: %d \n",x); // 30
      return 0;
}
```

Exemplo: função que troca de valores

O objetivo do programa abaixo é receber dois valores e, através de uma função, fazer a troca dos valores recebidos nas variáveis.

```
#include "stdio.h"

//função com passagem por referência
//a função tem dois ponteiros como parâmetros
void troca_valores(int *ptrx, int *ptry)
{
	int auxiliar;

	//auxiliar recebe o conteúdo apontado por ptrx
	auxiliar = *ptrx;

	//coloca o valor que está no local apontado por ptry em ptrx
	*ptrx = *ptry;

	//ptry recebe o valor armazenado em auxiliar
	*ptry = auxiliar;

	return;
}


int main(void)
{
	int a, b;
	printf("Digite o primeiro valor: ");
	scanf("%d", &a);

	printf("Digite o segundo valor: ");
	scanf("%d", &b);

	printf("Voce digitou os valores na seguinte ordem: %d e %d\n", a, b);

	/* Chamada da função que troca os valores das variáveis a e b.
	Observe como são passados os endereços nos argumentos. */
	troca_valores(&a, &b);

	//Exibindo os valores já trocados após a chamada da função
	printf("Os valores trocados sao: %d e %d\n", a, b);

}
```