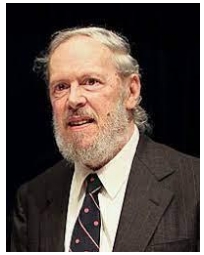
Módulo 1.

- Declarações e expressões em C
- Variáveis
- Testes e condições

Linguagem C e conceitos

Evolução da linguagem C



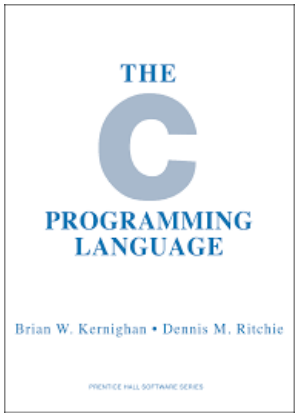
A linguagem C foi criada em 1972 por Dennis Ritchie† *2011* na Bell Labs.

Em 1978 é publicado por Kernighan & Ritchie o livro "The C Programming Language" (K&R C).

O American National Standards Institute elabora uma norma para a linguagem (finalizada em fins de 1988): ANSI C (também designado por C89).

A International Organization for Standardization (ISO) - C90 (aceitou o ANSI C, com pequenas modificações).

A linguagem C é relativamente estável, surgindo novas normas tipicamente de 10 em 10 anos aproximadamente: C99 e C11.



A estrutura básica de um programa

Na linguagem C um programa começa pela execução das instruções na função **`main`**

Todo o código a executar é colocado entre chavetas: **{ }**

As linhas de código entre um par de chavetas – **{ }** – designa-se por **Bloco**

Cada instrução dentro de um bloco de { } é terminada por um ponto-vírgula ;

A linguagem C é Case Sensite: main() difere de Main()

```
#include <stdio.h>

int main()
{
  printf ("Workshop C!\n");

  return (0);
}
```

Exemplo de um programa muito simples em C

Comentários

Comentários em bloco devem ser, preferencialmente, feitos com /* e */

Eventualmente, utilizando duas “barras” i.e., //

```c
#include <stdio.h>

int main(){

    // criar e inicializar uma variavel
    int x= -1; // atribuir valor negativo

    printf("x = %d  \n",x); // imprimir valor de x
/*
    Comentario de multiplas linhas ie, em bloco:
    vamos modificar o valor de "x"
    por exemplo: x = 5
*/
    x= 5;
    printf("x = %d  \n",x); // imprimir o novo valor de x

    return (1);
}
```

Evitar fazer comentarios com acentuacao e/ou carateres nao “internacionais”

Expressões simples

Operadores simples:

Operador	
*	// multiplicação
/	// divisão
+	// adição
-	// subtração
%	// resto da divisão inteira

Os operadores *, /, % têm precedência sobre a adição “+” e a subtração “-”.

Os parêntesis curvos “()” – e apenas estes – podem ser usados para agrupar termos.

Variáveis

Variáveis e armazenamento em C

A linguagem C permite armazenar valores em variáveis. Cada variável é identificada por um nome de variável.

O nome de uma variável: Sequência de letras ou dígitos. O primeiro caractere deve ser uma letra. O caractere _ é considerado uma letra.

A letras minúsculas são diferentes das maiúsculas. Logo a variável `max` é diferente de `Max`

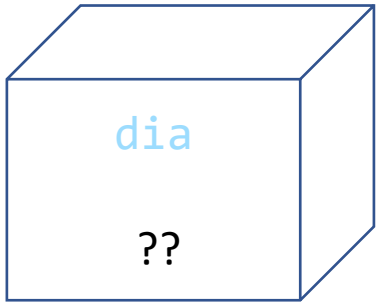
Exemplos: **int**, **float**, **while**, **for**.

Algumas palavras têm um significado especial na linguagem C pelo que palavras reservadas ou palavras chave – keywords– não podem ser utilizadas como nome de variáveis.

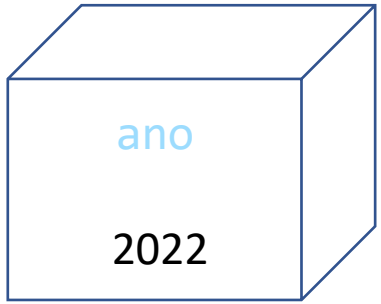
Conceito de variável em C

A definição de uma variável cria um contentor (memória) capaz de conter um valor de um dado tipo (o tipo **`int`** é utilizado para guardar valores inteiros):

```
int dia;
```




```
int ano = 2022;
```



Uma variável não pode ter um dos seguintes nomes:

```
3dias // comeca por um numero
$valor // contem um $
area circulo // contem um espaco
char // palavra reservada
```

Definição (tipos) e declaração de variáveis

Tipos simples, e mais usuais, de variável em C:

```
int k, num; /* numeros inteiros */

float x, zeta; /* números reais */

double gama; /* números reais */

char letra; /* caracter (tabela ASCII)*/
```

A função sizeof devolve o número de bytes que cada tipo ocupa na memória:

```
sizeof(int) == 4
sizeof(short int) == 2
sizeof(long int) == 8
sizeof(unsigned ) == 4
sizeof(unsigned short) == 2
sizeof(unsigned long) == 8
```

```
int k; // define k
k = 100; // atribui 100 a k (ou seja k <-- 100)

float num = 100.50;

int a, b, c, d;
a = b = c = d = 120; // d<--120; c<--d; b<--c; a<--b;
```

Vectores em C – breve introdução

Um "vector" (ou *array*) em C representa um conjunto de elementos consecutivos

```
tipo nome_variavel[numero_elementos];
```

tipo: pode ser *int, double, float, char*
numero_elementos: tem de ser inteiro > 0 ie, números naturais exceto 0.

```
int c[6];
float x[31];
```

Acesso aos elementos de um *array*

- O índice inteiro colocado entre [ ] indica o elemento do array.
- O índice do primeiro elemento (ou seja do elemento na primeira posição) é sempre 0 (zero). Por exemplo `c[0]`
- Neste caso, a variável `c` possui 6 elementos, sendo o primeiro `c[0]` e o último `c[5]`

Vectores em C – inicialização

Quando um *array* é definido (e caso não seja inicializado/preenchido), o valor dos seus elementos é indefinido/"*lixo*".

- Atribuindo todos os valores:

```
int X[4] = {3, -4, 5, 0};
char letras[3] = {'A','B','C'};
```

- Preenchendo tudo a 0 (zero):

```
int W[10] = { };
char letras[5] = { };
```

- Listando os primeiros, os restantes ficam com zero:

```
int Z[10] = {3, 4, -3};
char letras[5] = {'5','x'};
```

- Listando todos os valores que determinam o tamanho de um *array*:

```
int Y[] = {4, 0, -1, 0, 3, 5, -2, 8}; //8 elementos
char letras[] = {'a','b','c'};
```

Strings

Uma ***string*** é uma sequência de caracteres, devidamente terminada com **0**, armazenado numa tabela. O fim da sequência de carateres é assinalado pelo carácter NULL (ou seja, **\0**).

Em C as string são representadas usando aspas (ie., " ") enquanto os caracteres são representado entre plicas.

Exemplo da sintaxe de uma *string*:

```
char nome_da_string[5] = "DEEC"; //utilizar aspas
```

- Uma tabela de char **não** pode, *a-priori*, ser considerada uma string.
- Para o ser, uma tabela/array de chars **deve** de ter o carácter \0 (NULL) assinalando o fim da parte "útil" da tabela.

- O **'\0'** (NULL) tem o valor 0 (zero) ie, são 8 bits todos a 0.
- Contudo, strings são tabelas de char.

Strings – inicialização

A inicialização de strings segue uma sintaxe parecida com a inicialização de tabelas/arrays de caracteres.
Alguns exemplos de declaração e inicialização de ***strings***:

```
char S1[6] = {'D','E','E','C'};
char S2[6] = "DEEC";
char S3[ ] = "DEEC";
char S4[6] = ""; //todos caracteres são \0
```

Os valores dos elementos de cada ***string***:

```
S1[0]='D', S1[1]='E', S1[2]='E', S1[3]='C', S1[4]='\0', S1[5]='\0'

S2[0]='D', S2[1]='E', S2[2]='E', S2[3]='C', S2[4]='\0', S2[5]= ??

S3[0]='D', S3[1]='E', S3[2]='E', S3[3]='C', S3[4]='\0'
```

"lixo"

Neste exemplo, temos uma tabela de caracteres que **não** corresponde a uma ***string***:

```
char S[4] = {'D','E','E','C'};
```

A variável `S` **não** é uma *string* porque não possui o carácter terminador \0.

printf e scanf

A função printf – disponível na biblioteca *<stdio.h>*

A **função** printf é utilizada, neste exemplo, para enviar texto (entre aspas) para a consola (ou ecrã).

A **função** printf também pode ser utilizada para apresentar um caractere, uma sequência de caracteres, números ou resultados do cálculo de expressões numéricas.

main.c

```c
/******************************************************************************

                              Online C Compiler.
                Code, Compile, Run and Debug C program online.
Write your code in this editor and press "Run" button to compile and execute it.

*******************************************************************************/

#include <stdio.h>

int main()
{
    printf("Hello World");

    return 0;
}

```

A função printf – continuação

A função printf permite imprimir, com rigor e precisão, vários tipos ou formatos.

Exemplos de formatos: números inteiros, números em vírgula flutuante, etc.

Tipo	Formato (especificador)	Descrição
char	%c	Um único caractere
unsigned char	%c	Um único caractere
int	%d ou %i	Um inteiro base 10
int	%o	Um inteiro (base 8)
int	%x ou %X	Um inteiro (base 16)
short int	%hd ou %hi	Um short inteiro (base 10)
long int	%ld ou %li	Um long inteiro (base 10)
unsigned int	%u	Um inteiro positivo (base 10)
unsigned short int	%hu	Um short inteiro positivo (base 10)
unsigned long int	%lu	Um long inteiro positivo (base 10)
float	%f ou %e ou %g %F ou %E ou %G	Um número real precisão simples
double	%lf ou %le ou %lg %lF ou %lE ou %lG	Um número real precisão dupla

Operação sobre inteiros e reais

Na linguagem C qualquer operação que inclua uma variável do tipo float ou double i.e., números reais, obtém um resultado do tipo real.

```c
#include <stdio.h>

int main()
{
  float x = 13.0;
  double W = 5.00;
  int vp = 2, fp = 11;

  printf(" Resultado operacao = %f  \n", x/vp);
  printf(" Resultado operacao = %lf \n", vp*W);
  printf(" Resultado operacao = %d  \n", vp/(vp+fp)); //atencao!
  printf(" Resultado operacao = %f  \n", vp/W);

  printf(" Resultado com casting = %f \n", (float)vp/(vp+fp));
  return 0;
}
```

```
Resultado operacao = 6.500000
Resultado operacao = 10.000000
Resultado operacao = 0
Resultado operacao = 0.400000
Resultado com casting = 0.153846
```

A função scanf

A função scanf permite a leitura de valores.

De forma semelhante ao printf (escrita formatada), a função **scanf** funciona também com formatação i.e., %d ou %f ou %lf …

Exemplo de um programa com **scanf** para ler um inteiro

```
#include <stdio.h>
int main(){

    int n=0; //valor inicial de objetos

    printf(" Introduza o numero de objetos \n");

   //SCANF le o valor e armazena na variavel n
    scanf("%d", &n);

    printf(" Numero de objetos = %d  \n", n);


    return (1);
}
```

A função scanf

Na função **scanf**, depois de especificado o formato de leitura de todos os argumentos, estes devem ser colocados pela mesma ordem, precedidos de &. A string não deve conter outros caracteres além dos necessários para os formatos de leitura.

```c
/* Exemplo simples de utilizacao da SCANF */
#include <stdio.h>
int main() {

   float x=0.00, y=0.00, z=0.00; //coordenadas
   int n=1; //objeto

   printf("Digite o numero do objeto\n");
   scanf("%d", &n);

   printf("Digite as coordenadas x, y, e z \n");
   scanf("%f%f%f", &x,&y,&z );

  printf("Valores das coordenadas sao \n");
  printf("x = %1.2f; y= %1.2f; z= %1.2f \n", x, y, z);

   return (1);
}
```

```
Digite o numero do objeto
3
Digite as coordenadas x, y, e z
5
5.2
5.2222222
Valores das coordenadas sao
x = 5.00; y= 5.20; z= 5.22
```

A função scanf (leitura formatada)

Tipo	Formato (especificador)	Descrição
char	%c	Um único caractere
unsigned char	%c	Um único caractere
int	%d	Um inteiro (base 10)
int	%i	Um inteiro (base 10, 8 ou 16)
int	%o	Um inteiro (base 8)
int	%x ou %X	Um inteiro (base 16)
short int	%hd	Um short inteiro (base 10)
short int	%hi	Um short inteiro (base 10, 8 ou 16)
long int	%ld	Um long inteiro (base 10)
long int	%li	Um long inteiro (base 10, 8 ou 16)
unsigned int	%u	Um inteiro positivo (base 10)
unsigned short int	%hu	Um short inteiro positivo na base 10
unsigned long int	%lu	Um long inteiro positivo na base 10
float	%f ou %e ou %g %F ou %E ou %G	Um número real precisão simples
double	%lf ou %le ou %lg %lF ou %lE ou %lG	Um número real precisão dupla

Leitura de mais de um *char* com *scanf*

- Quando lemos caracteres (*char*) os chamados *whitespaces* (por exemplo: *<Enter>, space, Tab*), que são *inputs* do teclado, não são ignorados.
- Por outro lado, quando querem ler números (int, float ou double) a função *scanf* ignora os *whitespaces*.

Uma solução para este problema é acrescentar um **ESPAÇO** na função ***scanf***, por exemplo:

```
char c1, c2;
printf("Digite um caractere: \n");
scanf("%c",&c1);
printf("Digite outro caractere: \n");
scanf(" %c",&c2);


printf("\n Os caracteres sao: %c %c \n", c1, c2);
```

```
Digite um caractere:
a
Digite outro caractere:

 Os caracteres sao: a
```

Foi acrescentado um **ESPAÇO**

Controlo de fluxo: if-else

Verdadeiro (*true*) x Falso (*false*)

Em C, o número **0** (zero) corresponde ao valor lógico **Falso**.

Tudo o que é diferente de 0, inclusive números negativos, correspondem ao valor lógico **Verdadeiro**.

Exemplos:

0	**Falso**
-1	Verdadeiro
6	Verdadeiro
0.45	Verdadeiro
-1.1	Verdadeiro
-13	Verdadeiro

A linguagem C possui 4 tipos de dados ie, `char`, `int`, `float`, `double`.

Não existe nenhum tipo específico para representar valor lógicos ie, *true* ou false.

Contudo, em C o valor lógico *false* é representado por **0** e tudo que não seja igual a 0 representa o valor lógico *true*.

Operadores relacionais

Operador	Significado lógico	Exemplo	Interpretação
==	Igual	`x == y`	x é **igual** a y ?
!=	Diferente	`x != y`	x é **diferente** de y ?
<	Menor que	`x < y`	x é **menor** que y ?
<=	Menor ou igual	`x <= y`	x é **menor ou igual** a y ?
>	Maior que	`x > y`	x é **maior** que y ?
>=	Maior ou igual	`x >= y`	x é **maior ou igual** a y ?

Uma expressão que contenha um operador relacional devolve sempre o valor:
1 (true) ou 0 (false)

Operadores lógicos

Operador	Significado	Utilização
||	OU lógico	(x || y)
&&	E lógico	(x && y)
!	Negação lógica	! x

Operador ! // negacao	
X	!X
true	false
false	true

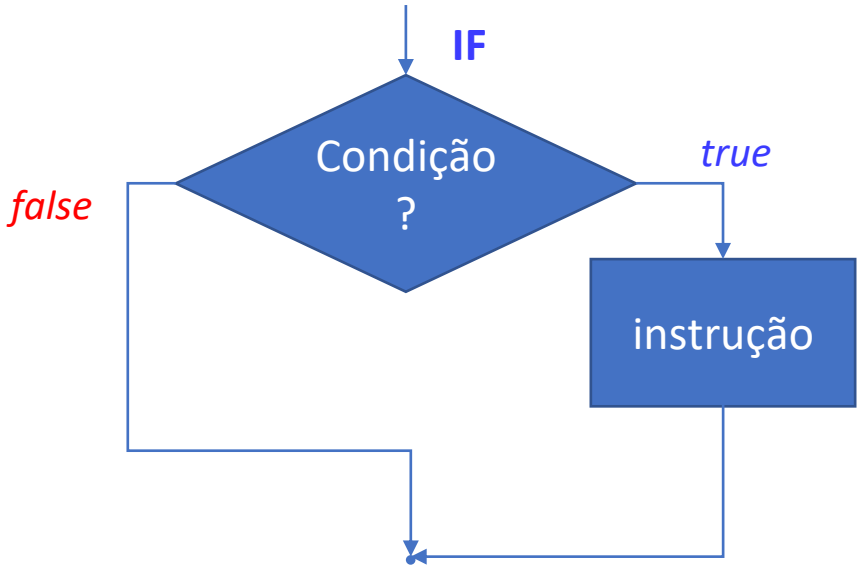
X	Y	(X || Y)	(X && Y)
false	false	false	false
false	true	true	false
true	false	true	false
true	true	true	true

Exemplos		
Variáveis, valores	Operador lógico, expressão	Resultado
x=5, y=0;	(x || y)	“1” ie, True
x=5, y=0;	(x && y)	“0” ie, Falso
x = -2;	!x	“0” ie, Falso

Instrução IF

```
if (condição lógica)
      instrução 1 ;
```

```
if (condição lógica){
      instrução 1 ;
      instrução 2 ;
      …
      instrução n ;
}
```



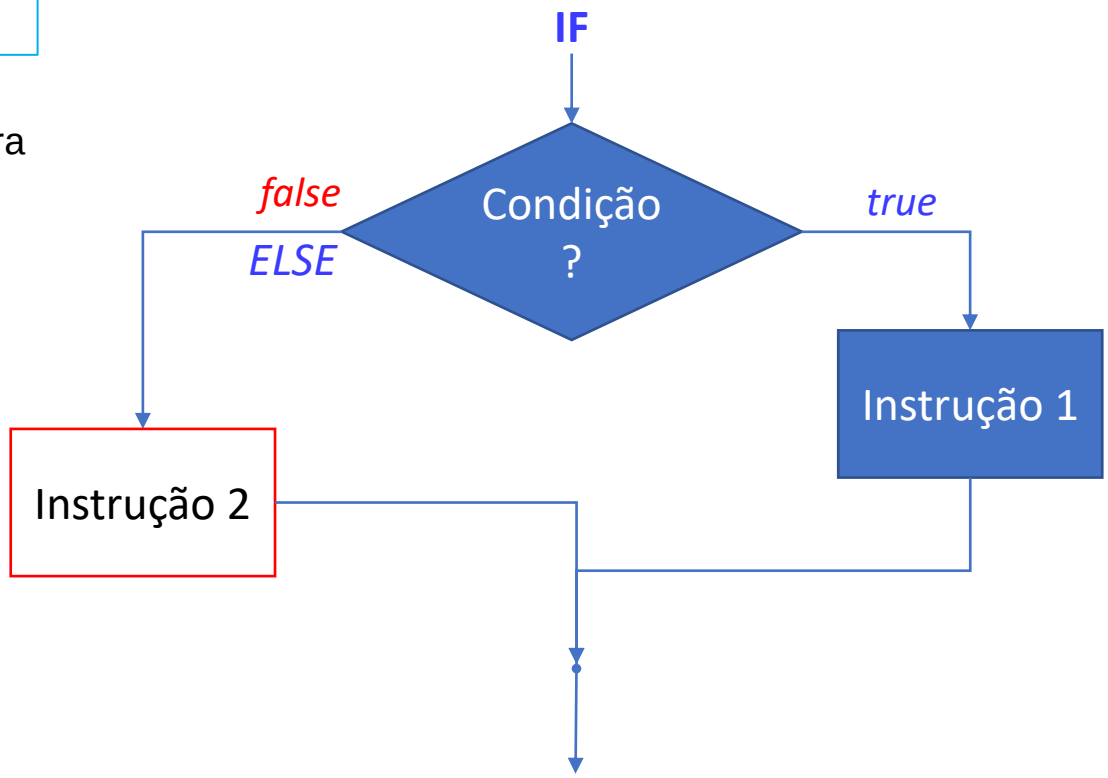

```
#include <stdio.h>

int main(){
  int previsao, real, v=0;

  previsao = real = 1;

  if ( previsao == real ){
     printf("Certo!\n", );
     v=1;
  }
  return (0);
}
```

Instrução IF-ELSE

```
if (condição lógica)
        instrução 1 ;
else
        instrução 2 ;
```

Podemos utilizar as chavetas { } para executar blocos de instruções

```
if ( condição )
        instrução 1;
else
        instrução 2;
```



Exemplo com IF - ELSE

```c
#include <stdio.h>

int main(){

  int previsao, real, v=0;

  previsao = 0;
  real = 1;

  if (previsao == real && real == 1){

     printf("Verdadeiro Positivo (VP)!\n");

     v=1;

  }else
     printf("Verdadeiro Negativo (VN)!\n");

  return (0);
}
```

IF-ELSE encadeados

```
#include <stdio.h>

int main(){
  int previsao, real, v=0;

  previsao = 0;
  real = 0;

  if (previsao == real){
        if (real == 1) printf("Verdadeiro Positivo (VP)!\n");
        else printf("Verdadeiro Negativo (VN)!\n");
  }else
     printf("Falso!\n");

  return (0);
}
```

// switch

Instrução *switch*

A instrução *switch* é uma alternativa eficaz aos "if-else encadeados", particularmente quando estes têm muitas opções.

A sintaxe da instrução switch:

```
switch (expressão)
{
        case   constante1: instrucoes1;
        case   constante2: instrucoes2;

        . . .
        case   constanten: instrucoesn;
        default: instrucoes;
}
```

- A **expressão** tem de ter um valor do tipo inteiro (*char*, *int*, *short* ou *long*).
- Se o valor da expressão for igual a alguma das constantes após um ***case***, então são executadas as instruções que se seguem a esse case; caso contrário são executadas as instruções após o default.
- O **default** é opcional, pelo que se este não existir e se nenhuma constante coincidir com expressão, então o switch termina.

Exemplo com SWITCH

```
#include <stdio.h>  //Exemplo com switch
int main()  {
  char opcao;
  printf("Qual a opcao [ A, B, C ou D] ? \n");
  scanf(" %c",&opcao);

  switch( opcao )
  {
        case    'A': printf(" Escolheu opcao A \n");
        case    'B': printf(" Escolheu opcao B \n");
        case    'C': printf(" Escolheu opcao C \n");
        case    'D': printf(" Escolheu opcao D \n");
        default: printf(" Nao escolheu uma opcao valida \n");
  }

  return (1);
}
```

Deverão verificar que se o valor **opcao** for igual a alguma das constantes após um case, então são executadas **todas** as instruções que se seguem a esse case.
- Ver próximo slide -

Instrução *switch* com *break*

No exemplo anterior, se o utilizador digitar 'A' <enter>, o resultado será:

```
Qual a opcao [ A, B, B ou D] ?
A
 Escolheu opcao A
 Escolheu opcao B
 Escolheu opcao C
 Escolheu opcao D
 Nao escolheu uma opcao valida


------------------
(program exited with code: 1)
Press any key to continue . . .
```

- Para impedir este efeito, deve-se usar a intrução **break**, que termina o switch assim que é encontrada.
- A última opção (o ***default*** ou o último ***case***) não precisa de break.
- Podemos utilizar ***break*** noutras situações – veremos mais adiante.

Exemplo com *switch* + *break*

```c
#include <stdio.h>  //Exemplo com switch + break

  int main() {
  char opcao;
  printf("Qual a opcao [ A, B, C ou D] ? \n");
  scanf(" %c",&opcao);

  switch( opcao )
  {
        case     'A': printf(" Escolheu opcao A \n"); break;
        case     'B': printf(" Escolheu opcao B \n"); break;
        case     'C': printf(" Escolheu opcao C \n"); break;
        case     'D': printf(" Escolheu opcao D \n"); break;
        default: printf(" Opcao invalida \n"); //ultimo caso nao precisa break
  }

  return (1);
}
```