

Algoritmos e Programação

1º Ano - 1º Semestre

2. Programação numa linguagem de alto nível (C) – Parte 1

Escola Superior de Tecnologia e Gestão de Viseu 2020-21

Agradecimentos a Francisco Morgado e Carlos Simões

2. Programação numa linguagem de alto nível (C) – Part 1

2.1 Introdução

2.2 Estrutura de um programa em linguagem C

2.3 Tipos de dados básicos

2.4 Testes e condições - expressões e operadores. Precedências

2.5 Estruturas de repetição

2.6 Funções

2.7 Tipos de dados estruturados: vectores e strings

2.8 Apontadores (Pointers)

2.9 Passagem de parâmetros de tipos estruturados

2.10 Estruturas

2.11 Memória dinâmica

2.1 Introdução

Algumas notas sobre a história da linguagem C

A linguagem C foi criada em 1972 nos *Bell Telephone Laboratories* por *Dennis Ritchie* e tinha por finalidade permitir a escrita de um sistema operativo (o Unix), utilizando uma linguagem de relativo alto nível, evitando assim o recurso ao *Assembly*.

A linguagem C, devido às suas capacidades e através da divulgação do Unix pelas Universidades dos USA, disseminou-se e tornou-se conhecida por todos os tipos de programadores, independentemente dos projectos em que estivessem envolvidos, o que levou ao desenvolvimento de diferentes versões. Em 1983 o *ANSI (American National Standards Institute)* formou um Comité para definição de um *standard* da linguagem C.

O nome da linguagem resulta da evolução de uma outra linguagem de programação (B) desenvolvida por Ken Thompson também nos laboratórios Bell

Algumas notas sobre a história da linguagem C (continuação)

C é uma linguagem particularmente estável, potente e flexível. A evolução das linguagens fez com que também esta evoluísse no sentido das Linguagens Orientadas por Objectos, dando origem a uma nova linguagem (C++), a qual mantém a sintaxe da linguagem C e permite um conjunto adicional de características.

Actualmente existe uma nova base alargada de trabalho para os programadores, baseada em C e C++, a linguagem Java.

Razões para programar em C

A maior parte das linguagens visa um determinado objectivo:

COBOL – processamento de registos;

FORTAN – cálculo científico;

LISP e PROLOG – vocacionadas para as áreas da Inteligência Artificial.

A linguagem C não se destina a nenhuma área de desenvolvimento em particular; é *general purpose*. Adapta-se ao desenvolvimento de quaisquer projectos, como sistemas operativos, compiladores, interfaces gráficas, processamento de registos, etc. É também muito utilizada para escrever os compiladores de outras linguagens.

Razões para programar em C

Existem compiladores de C para muitas arquitecturas/plataformas/sistemas operativos:

- Intel
- AMD
- PowerPC
- Windows
- Unix/Linux
- Mac OS
- dispositivos móveis
- calculadoras
- robots

Algumas características da linguagem C (1)

- ALTO NÍVEL** - o C é considerado uma linguagem de 3^a geração (tal como o COBOL, o BASIC, etc.), habitualmente denominadas de alto nível quando comparadas com o Assembly. Permite, no entanto, o acesso à maior parte das funcionalidades do Assembly, utilizando expressões/instruções de alto nível.
- RAPIDEZ** – performances semelhantes às da linguagem Assembly, usando instruções de alto nível.
- SIMPLICIDADE** – reduzido número de palavras reservadas, tipos de dados básicos e operadores e uma sintaxe simples; possui, contudo, poderosas bibliotecas adicionais com funções que permitem realizar inúmeras tarefas.
- PORTABILIDADE** – o código escrito numa máquina é transportável para outra máquina e compilado sem alterações (ou com um reduzido número de alterações).

Algumas características da linguagem C (2)

- MODULARIDADE** - o C permite o desenvolvimento modular de aplicações, facilitando a separação de projectos em módulos distintos e interdependentes.

- Criação de livrarias muito poderosas** - o facto do C conter um número reduzido de palavras chave, indica que as capacidades intrínsecas são limitadas (e.g., não tem instruções para realizar o input/output). No entanto, podem-se criar-se funções, incluídas em livrarias adicionais, que lhe permitem depois realizar todo o tipo de tarefas, desde a escrita de um carácter no ecrã, até ao processamento de strings, etc..

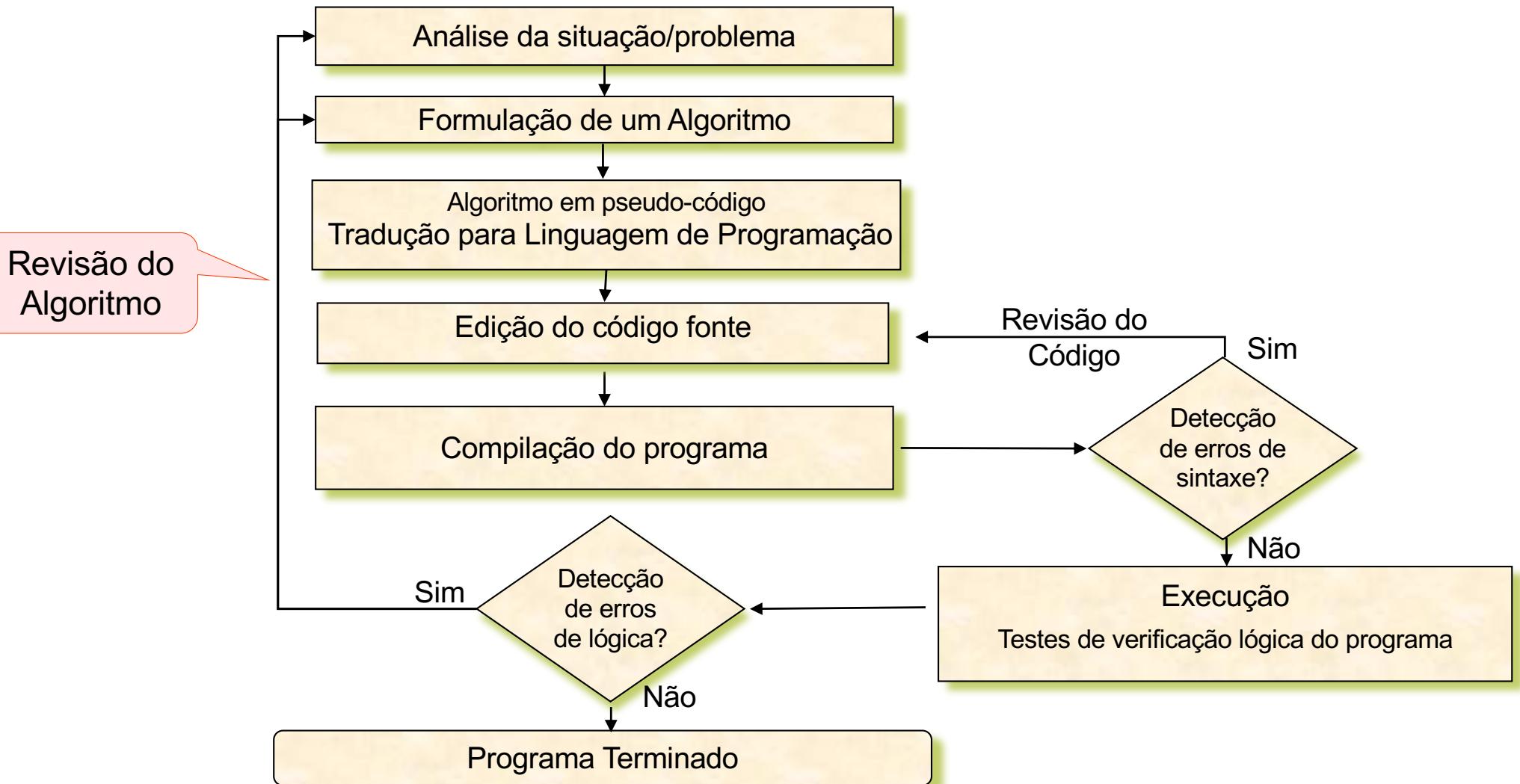
Lógica da Programação em C

A linguagem C reflete, de certo modo, a filosofia subjacente ao sistema operativo Unix, incorporando mesmo algumas das suas características (*case sensitive*(*), por exemplo): possui um largo conjunto de comandos, cada um dos quais é responsável pela realização de uma só tarefa. Caso se queira realizar uma tarefa complexa, recorre-se a um conjunto de comandos, encadeando-os através de estruturas de comunicação, para obter o resultado pretendido (como no Unix se encadeiam comandos, através de *pipes*).

A tática do “dividir para reinar” é regra na resolução de problemas. A complexidade subjacente a um programa complexo é maior do que a soma das complexidades dos vários programas simples que o constituem. Assim, surge a lógica da modularidade: um projecto deve ser dividido em módulos implementados de forma independente, os quais, por sua vez, devem ser divididos em componentes e cada componente poderá ainda ser subdividida, até que cada uma realize uma função apenas. Deste modo, torna-se mais fácil uma implementação correcta, reduzindo o tempo de desenvolvimento e de posterior manutenção.

- **(*) Case sensitive:** significa que um programa ou um compilador distingue letras maiúsculas e minúsculas. O termo vem do inglês e significa “sensível ao tamanho da letra”.
- **Todas as instruções de C são escritas com minúsculas**

Ciclo de desenvolvimento de uma aplicação



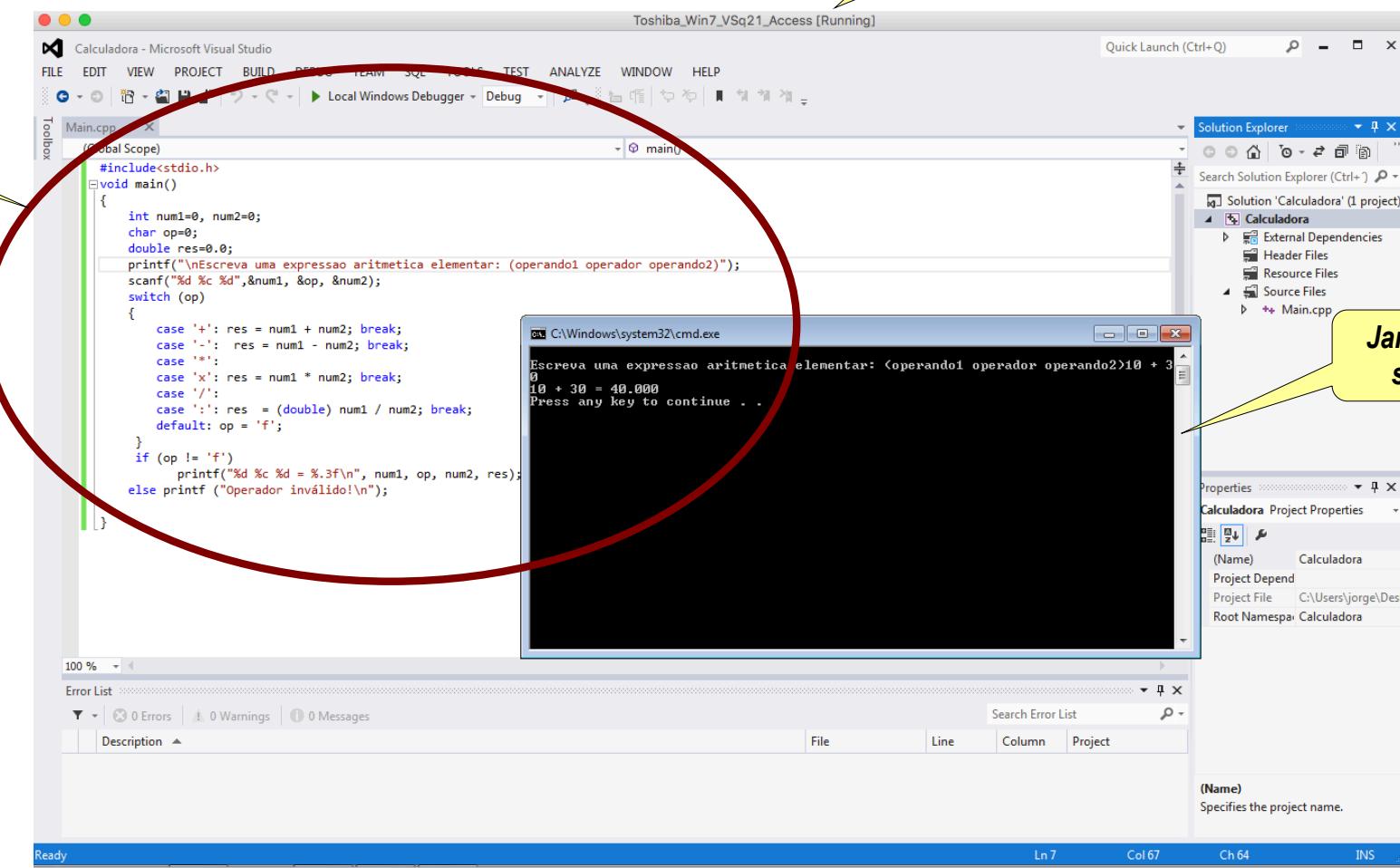
Ambiente Visual Studio

Ambiente de programação

Janela de edição de programas

Em ambiente Windows

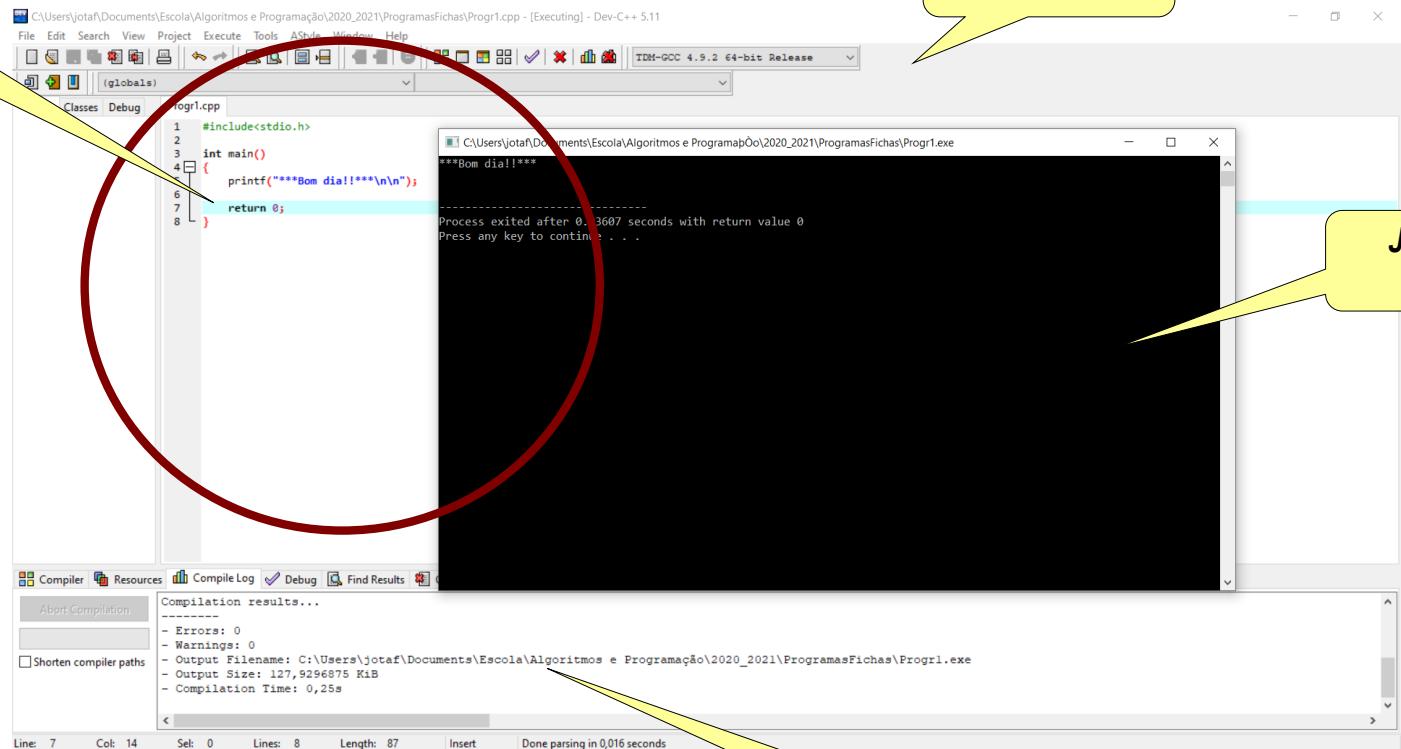
Janela de saída



Ambiente Dev-C++

Ambiente de programação

Janela de edição de programas



Em ambiente
Windows

Janela de
saída

Se houver erros,
aparecem nesta janela.

2.2 Estrutura de um programa em linguagem C

```
#include <biblioteca>
#define
tipos
variáveis globais
funções
```

- Quando o tipo é omitido, considera-se por omissão como sendo inteiro
- Sem tipo: *void*
- Nesta UC é obrigatório definir o tipo de dados, explicitamente

A ordem de colocação das funções é irrelevante

- Um programa em C é constituído por uma ou mais “**funções**”.
- A identificação das funções pode ser qualquer, excepto palavras reservadas da linguagem. Deve existir uma com a identificação **main** (*função principal*).

- São colocadas na função **main ()** todas as instruções que pretendemos sejam executadas
- Todo o código a executar é colocado entre {}

```
main()
{
}
```

Bloco

NOTA: A compilação deste programa devolve um Warning. Para que desapareça, colocar antes de **main** a palavra **void** ou **int** (dependendo do compilador).

Função Principal

```
void/int main()
{
    variáveis locais;
    instruções;
    // se int: return 0;
}
```

Exemplo:

```
#include <stdio.h>
void main()
{
    printf("Algoritmos e Programação");}
```

Em C cada instrução deve ser terminada com ponto e vírgula (;)

- ◆ A linguagem C, como já se referiu, não possui mecanismos de *input* e *output* incorporados (muito poucas palavras reservadas). Recorre a uma biblioteca de funções para fornecer esse tipo de serviços;
- ◆ `printf ("print+formatado")`: uma função que permite escrita no ecrã.

`#include <stdio.h>`

Directiva que indica ao pré-processador que deverá adicionar ao processo de compilação um ficheiro (`stdio.h`) existente (algures no disco) para que tenha acesso a um conjunto de informações sobre as funções a utilizar.

A extensão `h` significa que esses ficheiros não têm código, mas apenas cabeçalhos (*headers*) das funções que representam (designam-se habitualmente *header files*).

O ficheiro `stdio.h` permite o acesso a todas as funções de *input* e *output* normais.

O caractere especial \

O símbolo \ é usado para retirar o significado que tem um dado caractere

Exemplos:

- Tabulação horizontal: \t
- Tabulação vertical: \v
- Backspace: \b
- Representação do caractere \: \\
- fim de string: \0

- Caractere ' (plica): \'
- Caractere ?(ponto de interrogação): \?
- Mudança de linha: \n
- Caractere %: %%
- Caractere " (aspas): \"

```
#include <stdio.h>
void main()
{
    printf("Algoritmos e Programação \n");
}
```

*Em que difere
este programa
do anterior?*

```
#include <stdio.h>
void main()
{
    printf("\nAlg.\nne\nProgramação\n");
}
```

Notar que o comando \n (new line) gera uma mudança de linha

/* Comentários */

Um comentário em C é qualquer conjunto de caracteres compreendidos entre /* e */ ou que vem após //

2.3 Tipos de dados básicos

- Os tipos *char, int, float, double*
- Variáveis
- Atribuição
- Operadores numéricos - +, -, *, / e %
- Leitura de variáveis. Leitura e escrita de variáveis e expressões
- Formatos de leitura e escrita

Os tipos *char, int, float, double*

tipo	declaração	exemplo
caractere	char	char c;
inteiro	int	int si;
real	float	float x, y;
real de precisão dupla	double	double eps = 1.0e-5;
string	char identificador [dimensão]	char frase[80];
array	tipo identificador [dimensão]	int v[8]

NOTA

Na declaração de um inteiro podem ser usados 4 prefixos distintos, para melhor definição das características da variável:

- | | |
|-------------|---|
| ➤ short | - inteiro pequeno (2 Bytes) : ex. short int n1; // [-32 768, 32 767] |
| ➤ int | - inteiro (4 Bytes) : ex. int n2; // [-2 147 483 648, 2 147 483 647] = [-2 ³² / 2, 2 ³² -1] |
| ➤ long int | - inteiro longo (8 Bytes) : ex. long int n1; [-2 ⁶⁴ / 2, 2 ⁶⁴ -1] |
| ➤ /signed | - inteiro com sinal (por omissão, todos os inteiros são sinalizados) |
| ➤ /unsigned | - inteiro sem sinal (4 Bytes apenas números positivos) ex. unsigned int; // [0, 4 294 967 295] |

Para se saber qual a dimensão de uma variável (de qualquer tipo), o C disponibiliza o operador *sizeof*, cuja sintaxe é:

Exemplo:

sizeof expressão

OU

sizeof (tipo)

```
printf(" Tamanho de um Inteiro em Bytes : %d\n", sizeof(int));
```

Declaração de variáveis

Uma variável representa a área de memória, que contém um valor de um determinado tipo de dados. Para associar um determinado tipo de dados a uma dada variável, é necessário proceder-se à **declaração da variável**. A declaração da variável condiciona o tipo de dados que esta pode conter, durante a execução de um programa.

A declaração de uma variável é feita colocando antes do seu identificador a palavra reservada que representa o seu tipo.

Sintaxe da declaração de variáveis

```
tipoDeDados IdentificadorDeVariável [ , IdentificadorDeVariável ];  
...  
tipoDeDados IdentificadorDeVariável [ , IdentificadorDeVariável ];
```

Pode conter diversas variáveis do mesmo tipo, separadas por vírgula

Nome pelo qual passa a ser conhecida

Fim de instrução (;)

Regras a ter em conta na declaração de variáveis

- ◆ O identificador (nome) de uma variável pode ser constituído por letras do alfabeto e (maiúsculas ou minúsculas), dígitos e ainda pelo caractere *underscore* (_), mas deve **começar sempre por uma letra**.
- ◆ O identificador de uma variável **não pode**:
 - conter espaços e operadores;
 - ser igual a uma palavra reservada.
- ◆ O número máximo de caracteres de um identificador depende do compilador, mas usualmente é superior a 32.
- ◆ Existe distinção entre letras maiúsculas ou minúsculas na escrita de identificadores.
- ◆ A grande maioria dos compiladores não aceita caracteres acentuados.

Algumas recomendações

- ◆ O nome escolhido para uma variável deve sugerir qual a informação que armazena.
- ◆ A primeira letra do identificador deve ser minúscula.
- ◆ O nome de uma variável não deve ser todo escrito em letras maiúsculas: os programadores de C usam tradicionalmente maiúsculas para identificar constantes.
- ◆ Quando um identificador usar mais do que uma palavra, usar a diferença entre maiúsculas e minúsculas para as ligar (ex. totalDeVendas).

“Localidade” das variáveis

A declaração de variáveis não tem que ser feita no início (isto é, numa parte declarativa, a anteceder a operativa): pode ser feita em qualquer ponto do programa.

No entanto, é uma boa prática de programação, declarar as variáveis na parte declarativa.

Uma variável só passa a ser visível após a sua declaração.

As variáveis declaradas num bloco só são visíveis nesse bloco (geralmente um bloco é delimitado por chavetas).

Atribuição

Em C o operador de atribuição é constituído por igual “=”. Tal como o símbolo utilizado na descrição algorítmica (“ \leftarrow ”), também este tem o significado “*toma o valor de*”.

A instrução em pseudo-linguagem $i \leftarrow 3$ seria codificada em C através da instrução `i=3`.

NOTAS

1. Quando uma variável é declarada, fica sempre com um valor que resulta do estado aleatório dos bits que a constituem. Isto implica que o programador é responsável por inicializar uma variável antes de a “usar” ou podem ocorrer coisas estranhas!!!
2. Uma variável pode ser automaticamente inicializada quando se faz a sua declaração.

Exemplo: `int a = 10, b, c = -12, d;`

Atribuição

3. As instruções

```
a = 8;  
b = 8;  
c = 8;
```

equivalem à instrução

```
a = b = c = 8;
```

Pode ser. Porque em C, sempre que se faz uma atribuição, o valor do atributo é devolvido, como se fosse uma função, podendo ser utilizado por outras expressões ou variáveis.

4. Depois de executadas

as instruções:

```
x=1;  
y=2;  
z=3;  
x=y=z=4;
```



- a) Todas as variáveis ficam com 4;
- b) x fica com 2; y com 3 e z com 4;

Qual está correcta?

Perigo!!

Qual é o valor de x, y e z?

Havendo várias atribuições consecutivas, estas são realizadas da direita para a esquerda

Operadores

■ Operadores aritméticos

□ Binários

- Adição +
- Subtracção -
- Divisão /
- Multiplicação *
- Resto da divisão inteira %

□ Unários

- Incremento ++
- Decremento --

■ Operadores lógicos

■ Binários

- E &&
- OU ||

■ Unário

- Negação !

■ Operadores relacionais (binários)

- Maior >
- Menor <
- Maior ou igual >=
- Menor ou igual <=
- Igualdade ==
- Diferente !=

NOTA

Qualquer operação entre inteiros devolve um inteiro

Leitura de variáveis. Leitura e escrita de variáveis e expressões

scanf - leitura de variáveis

printf - escrita de variáveis e expressões

Para ler qualquer variável do tipo *int*, *char*, *float* ou *double* utilizando a função *scanf*, precede-se cada variável de &.

A *string* enviada para a função *scanf* não deve conter outros caracteres que não sejam os indicadores de formatos.

Formato de leitura e escrita de um inteiro nas funções *scanf* e *printf*: %d

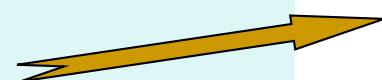
Exemplo ...

- printf("Escreva dois numeros: ");
- scanf("%d %d", &i1, &i2);
- printf("\nA soma dos valores lidos, %d e %d, é %d\n", i1, i2, i1+i2);
- ...



Formatos na instrução printf

- printf (str)
 - printf("Bom dia")
- printf (controlo, arg1, arg2,...)
 - controlo:
 - % **esp** tipoConversão
 - esp pode ser:



- As **flags** (por qualquer ordem):
 - - (**sinal menos**) - ajuste à esquerda
 - + (**sinal mais**) – mostra sempre o número com sinal
 - **Espaço** - se o primeiro caractere não for o sinal será mostrado um espaço em branco no início do número
 - 0 (**zero**) – preenche com zeros antes do número
- **número de largura mínima**
- . (ponto) **número de largura máxima (precisão)**
- h ou l - se o inteiro for para mostrar como short (h) ou long (l).

➤ tipoConversão pode ser:

- 
- d conversão para decimal
 - o conversão para octal sem sinal
 - x conversão para hexadecimal sem sinal
 - u conversão para decimal sem sinal
 - c conversão para **caractere**
 - s conversão para **string**
 - e conversão para o formato [-]m.nnnnnnE[±]xx (notação científica)
 - f conversão para **float**
 - g o mais curto entre %e e %f; zeros não significativos não são mostrados
 - % mostra o caractere % e não um argumento

Exemplos e notas sobre leitura e escrita de variáveis e expressões

O formato de leitura e escrita de variáveis inteiras **short** e **long** nas funções **scanf** e **printf** deve ser precedido dos prefixos **h** (short) e **l** (long)

Exemplo

```
...
• short int idade; /* ou short idade*/
• long x;
printf("Qual a idade? "); scanf("%hd",&idade);
printf("Qual o montante a depositar? "); scanf("%ld",&x);
printf("\nUma pessoa de %hd anos depositou %ld euros\n",idade, x);
• ...
```

• O formato de leitura e escrita de um inteiro sem sinal nas funções **scanf** e **printf** é **%u** em vez de **%d**

• O formato de leitura e escrita para números reais é **%f**

• Pode especificar-se o nº mínimo de caracteres com que é escrito um valor e o nº de casas decimais:

printf(" %10.3f -> número com 10 caracteres e 3 casas decimais \n",n);

• O total de caracteres pode omitir-se:

printf(" %.3f -> número com 3 casas decimais \n",n);

• O formato de escrita de reais em notação científica é **%e** (8.9e+3) ou **%E** (8.9E+3)

Exemplos:

■ Programa 1

```
#include <stdio.h>
void main()
{
    printf("\nPrimeiro Programa!\n");
}
```

■ Programa 2

```
#include <stdio.h>
void main()
{
    int x;
    printf("\nInsira um valor: ");
    scanf("%d", &x);
    printf("\nO valor inserido foi %d", x);
}
```

Nota

O espaço em branco dentro de um `scanf` (ex: `scanf(" %c", &ch)`) indica a esta função que ignore todos os *espaços em branco, new lines e tabs* que encontrar

Outra forma de limpar todos os caracteres que existam no *buffer* do teclado: usar a função `fflush(stdin)`

O formato de leitura e escrita usando `scanf` e `printf`, respectivamente, de uma string é `%s`

Exemplo:

- ...
- `char nome[50];`
- `printf('Insira o nome: '); scanf ("%s", nome);`
- `printf("\nNome lido: %s\n", nome);`
- ...

Observações sobre a função `scanf` na leitura de strings

- As variáveis que recebem a string NUNCA são precedidas de &
- A função `scanf` realiza a leitura de uma única palavra

Caracteres

Uma variável do tipo *char* permite armazenar um único caractere

Um *char* é armazenado num só *byte*, pelo que o número máximo de caracteres representáveis é 256

Verifique!

A representação de caracteres em C faz-se usando plica(') E NÃO aspas ("")

Exemplo: char letraA = 'A', letraZ = 'Z', nl = '\n';

O formato de leitura e escrita para caracteres é %c

Determine o resultado!

Programa

```
#include <stdio.h>
void main()
{
    printf("%c%c%crender faze%cdo!\n",'\'\n','\'A','\'p','\'n');
```

getchar() versus scanf()

A leitura de caracteres pode ser realizada com a função `scanf()` ou com a função `getchar()`.
`getchar()` DESTINA-SE UNICAMENTE À LEITURA DE UM CARACTER.

A função `getchar()` é invocada sem qualquer parâmetro.

Ex.: letra = getchar();

Caracteres e inteiros

Ao contrário do que sucede na maioria das linguagens de programação, os caracteres em C não são mais que inteiros guardados num único byte. Algumas consequências (algo inesperadas):

`char letra;`

`letra = 'A';`

`letra = 65; /* Caracter cujo código ASCII é 65 */`

`letra = '\101'; /* Caracter cujo código ASCII escrito em octal é 101 */`

`letra = '\x41'; /* Caracter cujo código ASCII escrito em hexadecimal é 41 */`

Tabela ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr
0	0	000	NUL (null)	32	20	040	Space	64	40	100	Ø	96	60	140	~
1	1	001	SOH (start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX (start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX (end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT (end of transmission)	36	24	044	\$	68	44	104	D	100	64	144	d
5	5	005	ENQ (enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK (acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL (bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS (backspace)	40	28	050	(72	48	110	H	104	68	150	h
9	9	011	TAB (horizontal tab)	41	29	051)	73	49	111	I	105	69	151	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	B	013	VT (vertical tab)	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	D	015	CR (carriage return)	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	E	016	SO (shift out)	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	F	017	SI (shift in)	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE (data link escape)	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1 (device control 1)	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2 (device control 2)	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3 (device control 3)	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4 (device control 4)	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN (synchronous idle)	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB (end of trans. block)	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN (cancel)	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM (end of medium)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB (substitute)	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC (escape)	59	3B	073	:	91	5B	133	[123	7B	173	{
28	1C	034	FS (file separator)	60	3C	074	<	92	5C	134	\	124	7C	174	
29	1D	035	GS (group separator)	61	3D	075	=	93	5D	135]	125	7D	175	}
30	1E	036	RS (record separator)	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US (unit separator)	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

Source: www.LookupTables.com

2.4 Testes e condições - expressões e operadores. Precedências

- Condições lógicas
- Instrução *if* - *else*
- Precedências dos operadores
- Instrução *switch*
- Instrução *break*

Em C não existe um tipo de dados específico para guardar valores lógicos

O valor lógico FALSO é representado por 0 (zero)

Tudo o que for diferente de 0 (zero) representa o valor lógico verdadeiro

O valor 1 é apenas um dos valores possíveis para representar VERDADE

Uma expressão lógica tem como resultado o valor FALSO (0) ou o valor VERDADEIRO (1)

Em C o valor de uma variável pode ser aproveitado pelo programador como valor lógico, utilizando-o como falso (caso seja 0) ou verdadeiro (caso seja diferente de 0)

Uma alternativa é declarar o tipo bool

```
enum boolEnum {false, true} ;  
typedef enum boolEnum bool;  
  
bool estado = true;
```

Operadores relacionais

ATENÇÃO

Um erro frequente em programação é a troca do operador **==** pelo operador **=**

Operador	Exemplo	Significado
==	$a == b$	a é igual a b?
>	$a > b$	a é maior que b?
>=	$a >= b$	a é maior ou igual que b?
<	$a < b$	a é menor que b?
<=	$a <= b$	a é menor ou igual a b?
!=	$a != b$	a é diferente de b?

Uma expressão que contenha um operador relacional devolve sempre como resultado o valor lógico FALSO(0) ou VERDADEIRO(1)

Operadores lógicos

Operador	Exemplo	Significado
&&	$x > 0 \&\& x <= 10$	AND (E lógico)
 	$x == 0 x == 1$	OR (OU lógico)
!	$!(x == 1)$	NOT (negação lógica)

Os operadores ++ e --

operador	exemplo	equivalente
++	<code>x++; ou ++x;</code>	<code>x = x+1;</code>
--	<code>x--; ou --x;</code>	<code>x = x-1;</code>

➤ Diferença entre `++x` e `x++`

Usando o operador ++	<code>y = x++;</code>	<code>y = ++x;</code>
Equivalente, sem o operador ++	<code>y = x; x = x+1;</code>	<code>x = x+1; y = x;</code>

- Quando o operador de incremento ou decremento está **antes** da variável, esta é **operada antes de ser usada**.
- Quando o operador está **depois** da variável, esta é usada e **só depois incrementada ou decrementada**.

Determinar o resultado das seguintes instruções:

```
i = -1;  
printf ("%d\n", ++i);  
printf ("%d", i++);  
printf ("%d\n", ++i);
```

0
0
2

Atribuição composta

► Quando uma variável recebe um novo valor que depende do seu valor actual, pode evitarse a duplicação da escrita da variável à esquerda e à direita, colocando o operador imediatamente junto (à esquerda) da atribuição.

variável = variável operador expressão



variável operador= expressão

► Quando se utilizam atribuições compostas, é absolutamente necessário que o **operador** fique **imediatamente junto ao símbolo de atribuição (=)**

Exemplos

Exemplo	Significado
$x += 1;$	$x = x + 1;$
$y *= 3+2;$	$y = y * (3+2);$
$a -= b+2;$	$a = a - (b+2);$
$c /= 8+k;$	$c = c / (8+k);$
$m \%= 2;$	$m = m \% 2;$

Prioridade dos operadores

Maior prioridade



Operador	Associatividade
<code>++ --</code>	Direita -> Esquerda
<code>* / %!</code>	Esquerda -> Direita
<code>+ -</code>	Esquerda -> Direita
<code>< <=> >=</code>	Esquerda -> Direita
<code>== !=</code>	Esquerda -> Direita
<code>&&</code>	Esquerda -> Direita
<code> </code>	Esquerda -> Direita
<code>= += -= *= /=</code>	Direita -> Esquerda

Menor prioridade

Note que em C o operador `&&` tem maior precedência que o operador `||`

Em caso de dúvida sobre as precedências, colocar explicitamente parêntesis!

Operador condicional ?

O operador **?** é o único operador ternário do C (espera três argumentos)

SINTAXE

condição ? expressão1 : expressão2;

Funcionamento

- É avaliada a condição:
 - Se o resultado for Verdade, é devolvido o valor da expressão1;
 - Se o resultado for Falso, é devolvido o valor da expressão2.

Exemplo:

Cálculo de novo preço: se preço < 10, preço = preço *1.1, senão preço = preço * 1.05

preco = preco < 10 ? preco * 1.1 : preco *1.05;

A linha de código acima, poderia ser escrita de outras formas:

Ex. preco = preco * (preco < 10) ? 1.1 : 1.05;

Instrução if - else

SINTAXE

```
if (condição)
    instrução1;
[else instrução2;]
```

Decisão condicional
SIMPLES
ou
DUPLA

- A condição do **if** tem de estar sempre dentro de parêntesis
- Tanto a **instrução1** como a **instrução2** são seguidas de ponto e vírgula (;

EXEMPLO

```
...
printf("Escreva um numero: ");
scanf("%d", &n);
if (n)
    printf("%d não é zero!\n",n);
else
    printf("%d é zero!\n",n);
...
```

Observação: Embora pareçam semelhantes, a instrução **if – else** e o operador **? não são iguais:**

a **instrução if – else** indica quais as instruções a executar, enquanto que o operador **?** devolve um resultado

EXEMPLO

Determinar o resultado das seguintes instruções:

```
i = -1;  
printf ("%d\n", ++i);  
if (++i)  
    printf ("%d", i++);
```

Verifica se `++i` é diferente de zero

0
1

Instrução switch

SINTAXE

```
switch (expressão)
{case constante1: instruções1;
 case constante2: instruções2;
 .....
 case constanten: instruçõesn;
 [default: instruções;]
}
```

Bem adaptado quando o número de possibilidades é elevado.

Decisão condicional
MÚLTIPLA

• Funcionamento

- Se o valor da expressão for igual a alguma das constantes que seguem os vários **case**, são executadas as instruções que seguem o **case** correspondente.
- Se o valor da expressão NÃO for igual a nenhuma das constantes apresentadas, são executadas as instruções que seguem **default**.
- O **default** é opcional. O que irá suceder, se não existir?

• Observações

1. Em cada **case** do **switch** só pode estar presente para avaliação uma constante do tipo *char*, *int* ou *long*.
2. Quando o valor da expressão é igual à constante de um dos cases, é executada a instrução associada a esse case bem como as instruções de todos os case seguintes (**default** incluído).
3. A instrução **break** permite parar a execução dentro de um **switch**, continuando o programa na instrução imediatamente a seguir ao término do **switch**.

Instrução **break**

A instrução **break** permite terminar um ciclo ou um **switch**

EXEMPLO – simulação de máquina de calcular (resultado num único printf)

```
...
printf("Escreva uma expressão aritmética elementar: (operando1 operador operando2)");
scanf("%d %c %d",&num1, &op, &num2);
switch (op)
{
    case '+': res = num1 + num2; break;
    case '-': res = num1 - num2; break;
    case '*':
    case 'x': res = num1 * num2; break;
    case '/':
    case ':': res = num1 / num2; break;
    default: op = 'f';
}
if (op != 'f')
    printf("%d %c %d = %.3f\n", num1, op, num2, res);
else printf ("Operador inválido!\n");
...
```

Testes e condições

EXEMPLO – cálculo do número de dias de um mês dado

```
void main()
{
    int ano, mes, numDias;
    printf("Indique um ano e um mês [1..12]: ");
    scanf("%d %d", &ano, &mes);
    switch (mes)
    {
        case 4:
        case 6:
        case 9:
        case 11: numDias = 30; break;
        case 2:
            if ( ((ano%4==0) && (ano%100 !=0)) || (ano%400 ==0) )
                numDias = 29;
            else numDias=28;
            break;
        default: numDias = 31;
    }
    printf("\nO mês %d do ano %d tem %d dias\n", mes, ano, numDias);
}
```