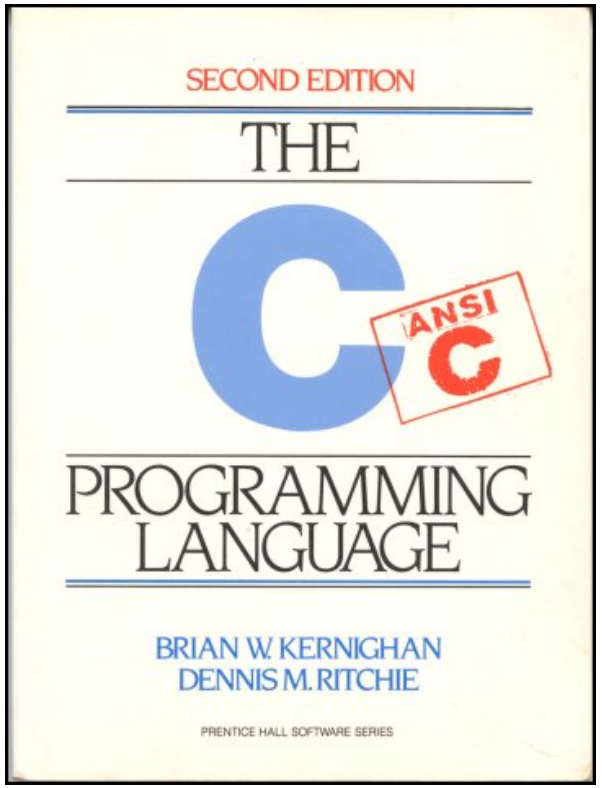
# Programação 2

Revisão C

Ana Paula Rocha
Luis Teixeira
**MIEEC 2017/2018**

# Linguagem C

- Origem
  - 1972, Bell Labs
  - K&R C "The C Programming Language", [Kernighan 1978]
  - ANSI C – standard em1989, X3.159-1989
- Linguagem imperativa (procedimental)

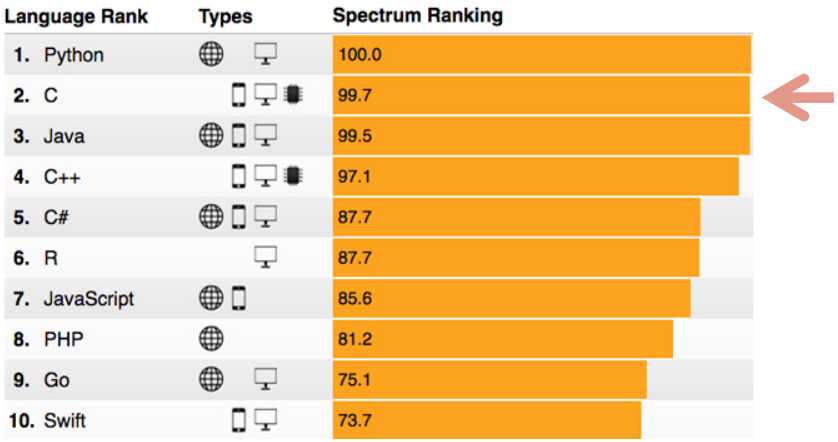


# Porquê linguagem C?

- Características:
  - código "portável" e eficiente
  - rapidez, estabilidade e disponibilidade quase universal
  - acesso a endereços específicos de hardware
  - flexibilidade de acesso a dados com diferentes formatos e requisitos
  - baixos requisitos para os recursos do sistema

- Linguagem de programação de sistemas
  - sistemas operativos (todos os principais)
  - sistemas embarcados

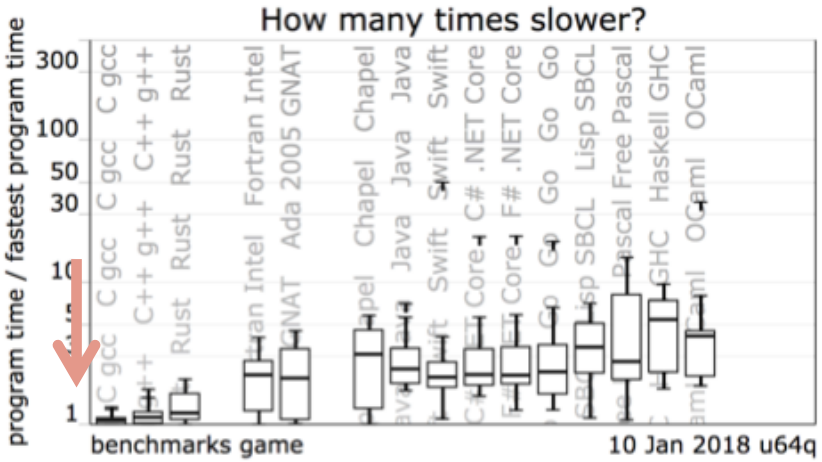
# Porquê linguagem C?



| Language Rank | Types | Spectrum Ranking |
|---|---|---|
| 1. Python | | 100.0 |
| 2. C | | 99.7 |
| 3. Java | | 99.5 |
| 4. C++ | | 97.1 |
| 5. C# | | 87.7 |
| 6. R | | 87.7 |
| 7. JavaScript | | 85.6 |
| 8. PHP | | 81.2 |
| 9. Go | | 75.1 |
| 10. Swift | | 73.7 |

- Principal linguagem de programação em projectos open-source [Google Code]
- Base das principais linguagens de programação modernas

https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages | Julho 2017




- Pequena camada de abstracção e baixo *overhead*
- Permite implementações eficientes de algoritmos e estruturas de dados

http://benchmarksgame.alioth.debian.org/u64q/which-programs-are-fastest.html | Janeiro 2018

# Programa em C

```
#include <stdio.h>
main()
{
    int expoente, n;
    float base, x=1;

    printf("Base: ");
    scanf("%f", &base);
    printf("Expoente: ");
    scanf("%d", &expoente);
    n=expoente;
    while (n > 0) {
        x=base*x;
        n=n-1;
    }

    printf("%f^%d = %f\n",
       base, expoente, x);
}
```

Declarar e inicializar

Ler dados

Processar

Escrever resultados

# Conceitos base Linguagem C

.: Todos estes conceitos foram estudados em Programação 1

.: Slides das aulas de Programação 1 no Moodle

- Dados
  - Armazenamento → Variáveis
  - Manipulação → Operadores
- Estruturas de controlo
  - Selecção
  - Repetição
- Vetores
  - Strings
- Funções
- Registos (ou estruturas)

# Variáveis

locais na memória (referenciados por um nome) que permitem armazenar valores

Declaração de variáveis

```
tipo     nome_da_variável;
tipo     nome_da_variável = 0;
tipo     nome_da_variável1, nome_da_variável2;
```

```
int x;          /* a variável x é do tipo int */
int s = 1;      /* variável s é inicializada com o valor 1 */
float  y, z;    /* duas váriáveis do tipo float */
```

Nomes de variáveis

- qualquer comprimento
- qualquer identificador válido (excepto palavras reservadas ou com digito no início)

xx <> xX <> Xx <> XX

# Leitura e escrita de dados

.: mais info Prog1 aula **3**

## printf()
escrita formatada de dados

```
printf("Hello World\n");
printf("Nome: %s %s\n", nome, apelido);
printf("soma = %d", s);
printf("media = %.2f", m);
```

## scanf()
leitura formatada de dados

```
scanf("%d", &valor);
scanf("%d %f %f", &n, &valor1, &valor2);
scanf(" %c", &letra);
scanf("%s %s", nome, apelido);
if(scanf("%s %d", nome, nota) == 2) {
    …
}
```

## Tipos de dados

.: mais info Prog1 aula **3**

- Números inteiros
  - char (8 bits / 1 byte)
  - int (16/32 bits)
  - short, long, unsigned são variantes de int
- Números reais
  - float (32 bits)
  - double é uma variante de float

# Operadores aritméticos

| | |
|---|---|
| + | adição |
| - | subtração |
| * | multiplicação |
| / | divisão |
| % | resto (módulo) |

operações entre inteiros

86 / 10 = 8

86 % 10 = 6

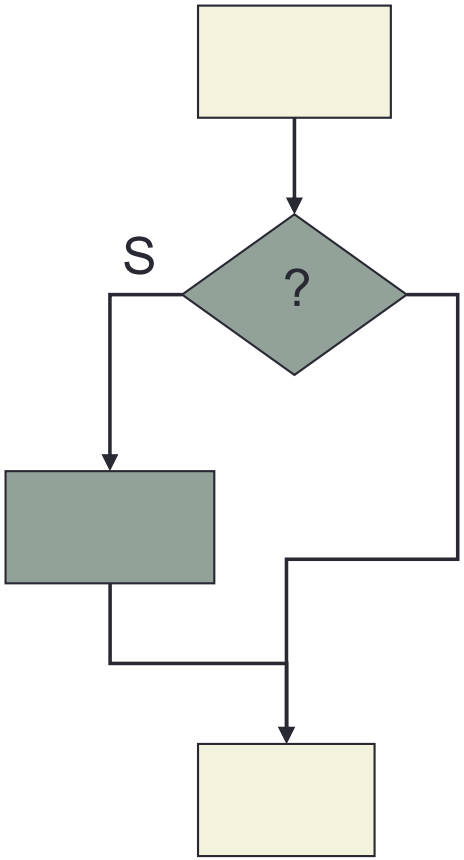
| Prioridade | Operador |
|---|---|
| 1 | -(unário) +(unário) |
| 2 | * / % |
| 3 | + - |

# Operadores lógicos

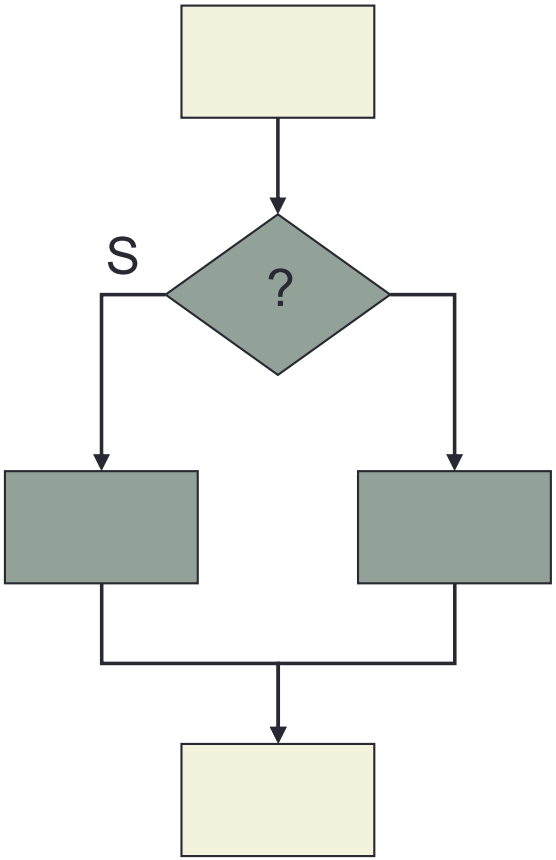
| | |
|---|---|
| < | menor que |
| <= | menor que ou igual |
| > | maior que |
| >= | maior que ou igual |
| == | igual |
| != | não igual |

| | |
|---|---|
| && | e |
| \|\| | ou |
| ! | negação (operador unário) |

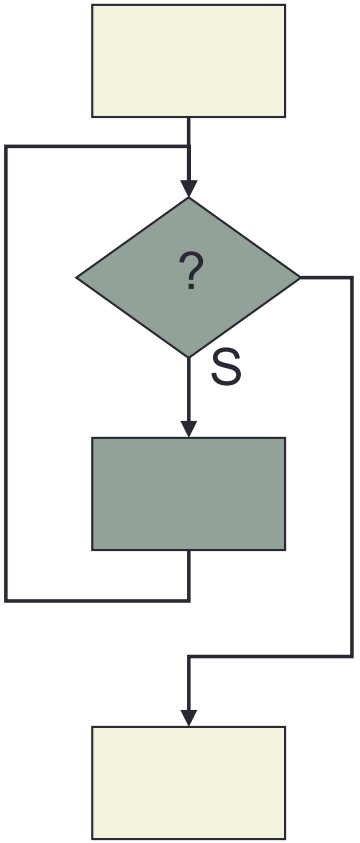
# Estruturas de controlo




if




if - else




while

# Estruturas de controlo

.: seleção
- if / if-else
- switch

.: repetição
- while
- do-while
- for

.: mais info Prog1 aulas **5-8**

## Exemplos

```
while(scanf("%d", &x) == 1) {
    if(x > 0) sum += x;
    n++;
}
if(n > 0)
    average = sum / n;
else {
    printf("can't compute average\n");
    average = 0;
}
```

```
for(i=0; i<10; i++) {
    scanf("%d", &x);
    if(x > 0) sum += x;
    n++;
}
```

# Apontadores

variáveis que armazenam endereços (de outras variáveis)

```
int *ptr;
// a variável ptr é um apontador (indicado pelo *) para uma variável
// do tipo int
// a variável ptr deverá guardar o endereço da zona de memória onde
// deverá estar um inteiro
```

## Utilização

```
int k; int *ptr;
ptr = &k;                    // Atribuição: uso do operador unário &
printf("%d\n",*ptr);         // Referência: uso do operador unário *
```

## Perigo

```
k = *ptr;        /* válido, mas só correcto se ptr estiver */
/* inicializado! Por exemplo: */
int i; ptr = &i;
k = *ptr;
```

# Apontadores

.: mais info Prog1 aula **10**

Quando um apontador é inicializado com um valor NULL, indica-se que "não aponta para nada". Podemos testar esta condição usando

```
if (ptr == NULL)
```

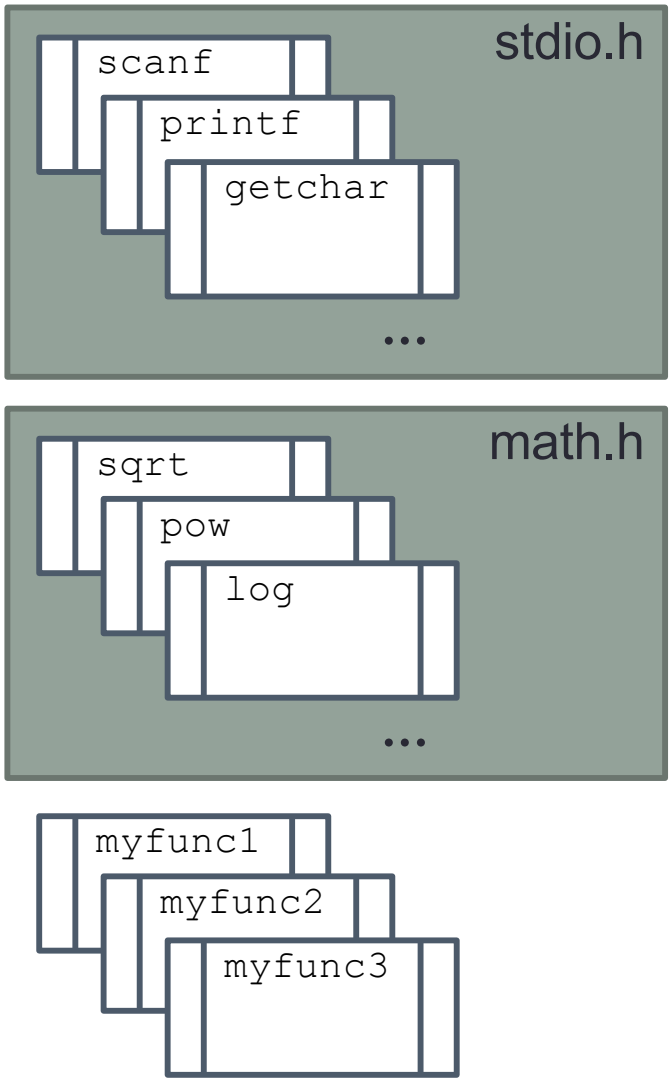
Há vários "tipos" de apontadores (mas todos ocupam o mesmo espaço)

```
int *ip;
char *cp; /* sizeof (ip) == sizeof(cp) */
```

Incrementar ou decrementar um apontador fá-lo avançar ou recuar para a variável seguinte ou anterior (por isso existem "tipos" de apontadores):

```
ip++; i--;
ip = ip + 2; ip = ip - 3;
```

# Funções




- A ideia de função é fundamental na construção de programas na linguagem C.
- A computação de um programa em C é dividida em funções, cada uma das quais realiza uma **tarefa pequena e bem definida**.
- As funções são normalmente desenhadas no contexto de um programa, mas devem ter sempre em vista ser **partilhadas** por diferentes programas.

# Funções

.: por omissão o tipo de retorno é int

.: main é uma função que quando termina, o programa também termina

Uma função declara-se:

```
tipo_de_retorno nome(lista_de_argumentos)
{
        lista_de_instruções
}
```

- A instrução:
  `return` expressão;
  permite retornar um valor ao programa que chamou a função.
- Se não houver a instrução `return`, o controlo é passado ao programa que chamou a função no final da última instrução antes de '`}`'.

# Funções

.: mais info Prog1
aulas **9, 13** e **18**

Passagem **por valor**:

```
sqr(a);
```

Indicamos à função ou ao procedimento o **valor** da variável / variáveis

A função ou o procedimento usa uma cópia e **não pode** escrever na variável / variáveis da função invocadora

Passagem **por referência**:

```
scanf("%d", &n);
```

Indicamos à função ou ao procedimento o **endereço** da variável / variáveis

A função ou o procedimento **pode** escrever na variável / variáveis da função invocadora

# Vetores

**Declaração:**

```
tipo_dos_elementos nome_vetor [quantidade];
```

**Declaração e inicialização:**

```
tipo_dos_elementos nome_vetor [quantidade]
 = {el_1, ..., el_n}; /* n <= quantidade */
```

**Exemplos:**

```
int notas[4];          /* notas é um vetor de 4 inteiros */

float salarios[12];    /* salarios é um vetor com 12 reais */

int notas[4] = {13, 10, 15, 12}; /* notas é um vetor com 4 valores */
                                 /* inteiros: 13, 10, 15 e 12 */

int notas[4] = {13, 10};         /* notas é um vetor com 4 valores */
                                 /* inteiros:  13, 10, 0 e 0 */
                     /* equivalente a int notas[4] = {13, 10, 0, 0}; */
```

# Vetores

.: mais info Prog1
aulas **15** e **16**

- Permitem representar numa só variável vários elementos, todos do mesmo tipo.
- São especialmente úteis quando queremos trabalhar com um número elevado de elementos
- Utilização:

```
/* i-ésimo elemento*/
nome_vector[i]
...
/* 1º elemento */
nome_vector[0]
...
/* último elemento */
nome_vector[quantidade-1]
```

# Strings

*Strings* são sequências de caracteres que compõem, por exemplo, qualquer texto.

Em C, são representadas como vetores de caracteres terminados com o carácter especial '\0'.

Este carácter, de código 0 (não confundir com o carácter '0'), é usado para marcar o fim da sequência.

Declaração:

```
char txt [8];
/* txt é um vetor de 8 caracteres */
```

Declaração e inicialização:

```
char msg[7] = {'V', 'i', 'v', 'a', ' ', '!', '\0'};
/* msg é um vetor com 7 caracteres, que */
/* constituem a sequência "Viva !" */
```

# Strings

Declaração e inicialização:

```
char msg[7] = {'V', 'i', 'v', 'a', ' ','!','\0'};

char msg[7] = "Viva !";
```

**Atenção**: o caráter terminador ocupa uma posição do vector!

Acesso aos elementos da sequência:

```
msg[i] /* acesso ao elemento índice i [0..6] de msg */

printf("%c", msg[2]); // o que imprime?
```

Impressão do conteúdo de uma sequência:

```
/* carater a carater: */
for(i = 0; i < 7; i++) printf("%c", msg[i]);

/* toda a sequência: */
printf("%s\n", msg);
```

# Strings

.: mais info Prog1 aula **17**

- Leitura de strings
  - `scanf("%s", nome)`
  - `gets(nome)`
  - `fgets(nome, max, stdin)`
- Escrita de strings
  - `printf("%s", nome)`
  - `puts(nome)`
- Operações
  - `int strlen(char *string)`
  - `char *strcpy(char *dest, char *src)`
  - `char *strcat(char *string1, char *string2)`
  - `double atof(char *string)`
  - `int atoi(char *string)`

# Registos

*Registo (struct em C) é um tipo estruturado de dados, construído a partir de outros tipos, que corresponde ao produto cartesiano de conjuntos.*

Declaração:

```
struct data {
        int ano, mes, dia;
};
```

Declaração e definição:

```
struct data {
        int ano, mes, dia;
}hoje;
```

Definição e utilização:

```
struct data ontem;
ontem.ano = 2010;
ontem.mes = 12;
ontem.dia = 16;
```

```
#include <stdio.h>
struct data {
        int ano;
        int mes;
        int dia;
};

main()
{
        struct data hoje;
        hoje.ano = 2010;
        hoje.mes = 12;
        hoje.dia = 17;
        printf("Hoje e'%d/%d/%d.\n",
hoje.ano, hoje.mes, hoje.dia );
}
```

# Registos

.: mais info Prog1
aula **19**

## Vetor num registo:

```
struct mes {
        int numero_de_dias;
        char nome[4];
};

struct mes este_mes;

este_mes.numero_de_dias = 30;
strcpy( este_mes.nome, "Nov");
printf("O mes e' %s\n", este_mes.nome);
```

## Vetor de registos:

```
struct data { int ano, mes, dia; };
struct data feriados[5];

feriados[1].mes = 12;

feriados[1].dia = 25;
```

# Ficheiros

.: mais info Prog1 aula **21**

- Tipos de ficheiros
  - Binários ou Texto (modo "b" ou "t")
- Acesso
  - `FILE *fopen(char *filename, char *mode)`
  - `int fclose(FILE *f)`
- Leitura/Escrita
  - `int fprintf(FILE *f, char *formato, ...)`
  - `int fscanf(FILE *f, char *formato, ...)`
  - `int fwrite(void *ptr, int size, int n, FILE *fich)`
  - `int fread(void *ptr, int size, int n, FILE *fich)`

```
printf("%s", CF>9 ? "done" : "again");
```