Fundamentos do C

CAP Diogo Silva

Table of contents

Tópicos 1
O primeiro programa 2
    Escrevi o código, e agora? 3
Componentes de um programa 4
    A diretiva *include* 4
    Funções 5
    Instruções 5
    I/O `printf` 5
Comentários 6
    documentação 6
Variáveis 7
    Variáveis > Tipos 8
    Variáveis > declaração 8
    Variáveis > atribuição 9
    Variáveis > printf 10
    Variáveis > `printf` de expressões 11
    O que é uma **expressão**? 11
    Variáveis > inicialização 12
    Exercício 12
I/O receber valores 13
    Exercício 14
Constantes 14
Identificadores 15

Tópicos

- O primeiro programa
- Componentes de um programa
- Comentários

- Variáveis
- I/O Receber Valores
- Constantes
- Identificadores

O primeiro programa

```c
#include <stdio.h>

int main(){
    printf("Hello world!");
    return 0;
}
```

O que está a acontecer?

Este programa simples escreve no terminal a frase “Hello world!”, sem as “.

```c
#include <stdio.h>

int main(){
    printf("Hello world!");
    return 0;
}
```

“Importa” código externo ao nosso programa, permitindo chamar a função printf, que não está aqui definida (linha 1).

```c
#include <stdio.h>

int main(){
    printf("Hello world!");
    return 0;
}
```

É a função principal do programa (linhas 3-6). Os programas em C começam sempre nesta função. Todos os programas têm uma função `main`.

```c
#include <stdio.h>

int main(){
    printf("Hello world!");
    return 0;
}
```

É uma instrução (linha 4) que chama a função `printf` da biblioteca `stdio.h` (que faz parte do próprio C), dando-lhe como input o literal de *string* `"Hello world!"`.

A função "imprime" no terminal o input recebido.

Todas as instruções no C acabam com `;`.

```c
#include <stdio.h>

int main(){
    printf("Hello world!");
    return 0;
}
```


Especifica o valor de retorno do nosso programa, neste caso 0 (linha 5).

Escrevi o código, e agora?

- preprocessamento
  - obedece a comandos que começam com # (diretivas)
  - funciona um pouco como um editor de texto, onde se adicionam ou modificam partes do código

- compilação
  - transformar o código C numa linguagem que o computador perceba = linguagem máquina
- *linking*
  - combina o output gerado pela compilação com outro código necessário à execução do programa (e.g. stdio.h para usar o printf)

Quando damos indicação ao nosso IDE para compilar e correr o código, estes 3 passos tipicamente acontecem de forma automática.

Componentes de um programa

```
#diretivas

int main(){
    instruções;
}
```

A diretiva *include*

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

Esta diretiva indica que o conteúdo da biblioteca `stdio.h` deve ser incluida no programa atual.

Funções

Funções são procedimentos ou subrotinas que executam uma determinado conjunto de instruções.

Cada função é uma determinada receita para um determinado comportamento ou funcionalidade.

Isto permite que o mesmo comportamento seja executado diversas vezes num programa sem o ter que implementar múltiplas vezes.

Instruções

Uma instrução é um comando que é executado quando o programa corre, e.g.

```
printf("Hello world!");
```

I/O `printf`

A capacidade de um programa comunicar com o mundo exterior é fundamental para que seja útil.

No nosso primeiro programa, essa comunicação foi feita com a função `printf` e o output foi apresentado num terminal.

Anteriormente foi referido que a função recebeu um *literal de string*, uma série sequencial de caracteres entre “ “.

As “ ” não são incluidas no que aparece no terminal. Se quisermos incluir ” no nosso output, é necessário escrever ”.

O `printf` não inclui automaticamente o fim de linha.

Para garantir que o `printf` seguinte não apresenta o seu conteúdo na mesma linha do anterior, é necessário adicionar `\n`.

```c
printf("Hello world!\n");
printf("Good morning, Vietnam!");
```

Comentários

O nosso primeiro programa não tem algo importante:

documentação

Os programas devem ter várias informações sobre o programa em si, e.g. autor, propósito, etc.

```c
/* Nome: hello_world.c*/
/* Autor: Diogo Silva*/
/* Proposito: Mostrar "Hello world!" na consola.*/
#include <stdio.h>

int main(){
    printf("Hello world!");
    return 0;
}
```

No C, escrevemos comentários entre `/* */`. Todo o texto entre estes símbolos é ignorado pelo comentário, i.e. não é interpretado como código.

```c
/*
Nome: hello_world.c
Autor: Diogo Silva
Proposito: Mostrar "Hello world!" na consola.
*/
#include <stdio.h>

int main(){
```

```
    printf("Hello world!");
    return 0;
}
```

Um comentário pode compreender várias linhas.

Também podemos escrever comentários de apenas uma linha com o símbolo `//`.

Tudo o que vem depois deste símbolo é interpretado como um comentário.

```
#include <stdio.h>

int main(){
    printf("Hello world!"); // escrever na consola
    return 0;
}
```

A legibilidade do programa aumenta bastante quando existem comentários que descrevem de forma sumária porções do código.

Variáveis

A maior parte dos programas executa uma série de cálculos.

Para isso, precisam de um mecanismo para guardar dados de forma temporária.

É para isso que servem as **variáveis**.

As **variáveis** são locais na memória onde é possível gravar algum dado, temporáriamente, durante a execução do programa.

Variáveis > Tipos

No C, todas as variáveis têm um **tipo** associado.

O C tem vários tipos de dados, mas para já usaremos apenas 2 tipos numéricos:

- int (inteiro)
- float (real, *floating point*)

Uma analogia aos tipos são os domínios na matemática, e.g. `int` pode ser comparado ao dominio dos números naturais.

Associado a um tipo, existe: - um conjunto de valores válidos (e.g. números inteiros positivos até 65.535) - operações válidas nos valores desse tipo (+ - * /)

Diferentes tipos incluem diferentes conjuntos de valores válidos.

O C tem vários `int` que incluem diferentes intervalos de valores possíveis e.g. típicamente, `int` inclui valores inteiros no intervalo `[-65536, 65535]`.

O `float` inclui valores muito maiores que um `int`, e contempla valores reais, e.g. 3.14.

Operações aritméticas em `float` podem ser mais lentas.

Variáveis > declaração

Quando queremos usar uma variável, esta tem de ser declarada.

```
int altura; // em cm
float massa; // em kg
```

`altura` é uma variável do tipo `int` e `massa` é uma variável do tipo `float`.

Também podemos declarar várias variáveis de um determinado tipo numa só linha.

```
int altura, idade; // cm, meses
float massa, lucro; // kg, €
```

Variáveis > atribuição

As variáveis recebem valores através da instrução de atribuição `=`.

```
int altura; // em cm
float massa; // em kg

altura = 180;
massa = 75.2;
```

`altura` tem agora o valor `180` e `massa` tem o valor `75.2`

`180` e `75.2` são constantes.

Depois das variáveis terem um valor atribuido, podem ser usadas no cálculo de outros valores.

```
int altura; // em cm
float massa, imc; // em kg

altura = 180;
massa = 75.2;
imc = massa / (altura / 100.0); //indice massa corporal
```

As variáveis só podem ser usadas depois de declaradas. Ver linha 3.

```
int altura; // em cm

massa = 75.2;

float massa, imc; // em kg
```

```
altura = 180;

imc = massa / (altura / 100.0); //indice massa corporal
```

Este código é inválido porque estamos a atribuir um valor à variável `massa`, que ainda não foi declarada.

Variáveis > printf

Se quisermos mostrar o valor de uma determinada variável, podemos usar novamente a função `printf`.

```
int altura; // em cm
float massa, imc; // em kg

altura = 180;
massa = 75.2;
printf("Altura: %d\n", altura);
```

O que significa o símbolo `%d`?

`%d` indica como é que o valor contido na variável `altura` deve ser interpretado.

Neste caso, `%d` indica que deve ser interpretado como um valor inteiro `int`.

O descritor escolhido deve ser coerente com o tipo da variável que se vai mostrar.

`altura` é do tipo `int`, logo usamos o descritor `%d`.

```
int altura; // em cm
float massa, imc; // em kg

altura = 180;
massa = 75.2;
imc = massa / (altura / 100.0); //indice massa corporal
```

```c
printf("Indice massa corporal: %f\n", imc);
```

O descritor `%f` é usado para valores do tipo `float`.

- `int` -> `%d`
- `float` -> `%f`

> **Importante**
>
> O C não impede um "desalinhamento" entre tipo e descritor - é responsabildiade do programador garantir a coerência.

Variáveis > `printf` de expressões

O cálculo do IMC na variável intermédia `imc` é desnecessário, uma vez que podemos inserir a **expressão** completa do cálculo na função `printf`.

```c
int altura; // em cm
float massa, imc; // em kg

altura = 180;
massa = 75.2;
printf("Indice massa corporal: %f\n", massa / (altura / 100.0));
```

O que é uma expressão?

- Uma expressão é algo que produz um valor concreto.
- **Uma expressão produz sempre um valor com um valor concreto**, mesmo quando variáveis de tipos diferentes são misturadas (conversão automática de tipos será abordada posteriormente.)
- Um valor de um determinado tipo pode sempre ser substituido por uma expressão do mesmo tipo.

Variáveis > inicialização

- Algumas variáveis são automaticamente inicializadas com o valor de 0 quando são declaradas, mas a maior parte não é.
- Uma variável que não tenha um valor por defeito e que não tenha sofrido nenhuma atribuição está **não inicializada**.

É importante perceber que o C permite que uma variável seja usada em cálculos, mesmo não tenha sido inicializada.

```
int altura; // em cm
float massa, imc; // em kg

massa = 75.2;
imc = massa / (altura / 100.0); //indice massa corporal
```

É **responsabilidade do programador saber que o programa pode ter um comportamento inválido**, porque a variável que não foi inicializada (não teve nenhuma atribuição) pode conter qualquer valor.

As variáveis podem ser inicializadas no momento da sua declaração.

```
int altura=180; // em cm
float massa=75.2, imc=0.0; // em kg
```

Exercício

- Conversão de Fahrenheit para Celsius.
- $C = \frac{5}{9}(F - 32)$

I/O receber valores

Já sabemos mostrar valores ao utilizador do programa, mas como receber dados?

Usamos a função `scanf`.

A função `scanf` funciona como o `printf`, mas na direção oposta.

```
scanf("%d", &a);
scanf("%f", &b);
```

Indicamos o formato dos dados que vamos receber e a variável onde os queremos guardar.

```
scanf("%d", &a);  // receber altura do utilizador
scanf("%f", &b);  // receber altura do utilizador
```

Assumindo que já foram declaradas, qual será o tipo das variáveis `a` e `b`?

Sabemos que `%d` funciona com valores `int`, logo `a` deverá ser do tipo `int`.

Na mesma linha, `b` deverá ser um `float`.

E o que significa o `&` antes do nome das variáveis?

```
scanf("%d", &a);
scanf("%f", &b);
```

É um operador que devolve o *endereço de memória* da variável.

Iremos explorar este operador em detalhe no futuro. Até lá, saibam apenas que usamos quase sempre o `&` antes do nome da variável.

Exercício

Conversão de ºF para ºC.

Alterar o exercício anterior para receber os valores do utilizador.

Constantes

O que faz este programa?

```c
#include <stdio.h>

int main(){
    float r;
    printf("Insira o raio do circulo[cm]:");
    scanf("%f", &r);

    printf("Perimetro do circulo: %f", 2 * 3.1415 * r);
    printf("Area do circulo: %f", 3.1415 * r * r);
    return 0;
}
```

Neste programa o valor 3.1415, o valor do repete-se.

Seria útil se pudéssemos fazer referência a este valor com um identificador em todo o programa.

O C tem uma diretiva que nos permite fazer exatamente isso.

```c
#include <stdio.h>
#define PI 3.1415

int main(){
    //...

    printf("Perimetro do circulo: %f", 2 * PI * r);
```

```
    printf("Area do circulo: %f", PI * r * r);
    return 0;
}
```

A diretiva `#define` permite-nos definir constantes que, durante o préprocessamento, são substituidas pelo valor especificado.

Por convenção, os nomes das constantes são sempre em letras maíusculas e os nomes das variáveis são em minúsculas.

Identificadores

Os nomes que escolhemos para as nossas variáveis, funções, etc. designam-se por *identificadores* e existem regras que devem ser seguidas.

Exemplos de identificadores válidos

```
times10 proximo_numero _altura alturaMAX
```

Exemplos de identificadores inválidos

```
10times proximo-numero
```

O caracter `-` é inválido, mas `_` é aceite.

Os identificadores são sensíveis a letras maísculas e minúsculas, e.g. `alturaMAX` e `alturamax` seriam 2 identificadores distintos num programa.

Existe um conjunto de palavras-chave que não podem ser usadas. Estas correspondem a elementos intrínsecos ao C, como o nome dos tipos básicos e dos ciclos, e.g. `int`, `while`.

Outros exemplos:

```
int float double char void long short typedef
if else switch default
while do for continue break
struct enum union
```

> **Importante**
>
> O uso do nome de funções frequentemente usadas e pertencentes à biblioteca *standard* do C também é de evitar, e.g. `printf`, `scanf`, ...