2ª Aula - Os Meus Primeiros Programas em C

Programação
Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;
- Cada uma dessas **tarefas** é realizada por um **conjunto de instruções** que, por sua vez, se podem agrupar em '**funções**';

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;
- Cada uma dessas **tarefas** é realizada por um **conjunto de instruções** que, por sua vez, se podem agrupar em '**funções**';
- Assim, do ponto de vista do '**C**', essencialmente têm-se **funções**;

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;
- Cada uma dessas **tarefas** é realizada por um **conjunto de instruções** que, por sua vez, se podem agrupar em '**funções**';
- Assim, do ponto de vista do '**C**', essencialmente têm-se **funções**;
- Por construção, em '**C**', a **primeira função** a ser executada designa-se por '**main**';

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;
- Cada uma dessas **tarefas** é realizada por um **conjunto de instruções** que, por sua vez, se podem agrupar em '**funções**';
- Assim, do ponto de vista do '**C**', essencialmente têm-se **funções**;
- Por construção, em '**C**', a **primeira função** a ser executada designa-se por '**main**';
- As **funções** podem ter **argumentos** (**valores** que recebem) que ficam **entre parêntesis**,

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;
- Cada uma dessas **tarefas** é realizada por um **conjunto de instruções** que, por sua vez, se podem agrupar em '**funções**';
- Assim, do ponto de vista do '**C**', essencialmente têm-se **funções**;
- Por construção, em '**C**', a **primeira função** a ser executada designa-se por '**main**';
- As **funções** podem ter **argumentos** (**valores** que recebem) que ficam **entre parêntesis**, e também podem **retornar um valor**;

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;
- Cada uma dessas **tarefas** é realizada por um **conjunto de instruções** que, por sua vez, se podem agrupar em '**funções**';
- Assim, do ponto de vista do '**C**', essencialmente têm-se **funções**;
- Por construção, em '**C**', a **primeira função** a ser executada designa-se por '**main**';
- As **funções** podem ter **argumentos** (**valores** que recebem) que ficam **entre parêntesis**, e também podem **retornar um valor**;
- As **funções** são constituídas por um **bloco de instruções** que fica entre chavetas { **...** }

Estrutura básica do 'C'

- Um programa em '**C**' é uma **coleção** de **tarefas** executadas numa dada sequência;
- Cada uma dessas **tarefas** é realizada por um **conjunto de instruções** que, por sua vez, se podem agrupar em '**funções**';
- Assim, do ponto de vista do '**C**', essencialmente têm-se **funções**;
- Por construção, em '**C**', a **primeira função** a ser executada designa-se por '**main**';
- As **funções** podem ter **argumentos** (**valores** que recebem) que ficam **entre parêntesis**, e também podem **retornar um valor**;
- As **funções** são constituídas por um **bloco de instruções** que fica entre chavetas { **...** }
- As **instruções**, contidas nesse bloco, são executadas sequencialmente e cada uma delas termina por '**;**'.

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;

main

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };

main

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };

```
main
{



}
```

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
main
{



}
```

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{



}
```

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;
- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;

```
int main ()
{



}
```

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;
- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;

```
int main ()
{
   printf ("Ola a todos!")

}
```

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{
   printf ("Ola a todos!")

}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{
   printf ("Ola a todos!");

}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{
   printf ("Ola a todos!");

}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução
- Dar a indicação de mudar de linha;

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{
   printf ("Ola a todos!\n");

}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução
- Dar a indicação de mudar de linha;

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{
   printf ("Ola a todos!\n");

}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução
- Dar a indicação de mudar de linha;
- Pôr o retorno da função.

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{
   printf ("Ola a todos!\n");
   return 0;
}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução
- Dar a indicação de mudar de linha;
- Pôr o retorno da função.

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
int main ()
{
   printf ("Ola a todos!\n");
   return 0;
}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução
- Dar a indicação de mudar de linha;
- Pôr o retorno da função.
- Incluir as definições das funções de *input* e *output*

Dizer: Olá a todos! (I) ('Prog01_01.c')

- Construir um programa em '**C**' é, antes de mais, escrever a função **main** e as funções a ela ligadas;
- Como se viu, uma função tem um bloco de instruções entre chavetas { **...** };
- A função **main** é um função (obrigatoriamente inteira), logo é necessário acrescentar **int** e **()** a seguir ao nome da função;

```
#include <stdio.h>
int main ()
{
   printf ("Ola a todos!\n");
   return 0;
}
```

- Dar a instrução de escrever: inserir a função **printf** com o que queremos escrever entre aspas;
- Pôr o '**;**' no final da instrução
- Dar a indicação de mudar de linha;
- Pôr o retorno da função.
- Incluir as definições das funções de *input* e *output*

Dizer: Olá a todos! (II)

- Acabámos de construir o programa para dizer '**Olá a todos!**'. Que fazemos com ele?

Dizer: Olá a todos! (II)

- Acabámos de construir o programa para dizer '**Olá a todos!**'. Que fazemos com ele?
- Este programa tem de ser escrito num ficheiro que deverá ter a extensão '**.c**'. Chamemos-lhe, por exemplo, **Prog01_01.c**. Para o escrever podemos usar um editor, por exemplo, o **emacs**;

Dizer: Olá a todos! (II)

- Acabámos de construir o programa para dizer '**Olá a todos!**'. Que fazemos com ele?
- Este programa tem de ser escrito num ficheiro que deverá ter a extensão '**.c**'. Chamemos-lhe, por exemplo, **Prog01_01.c**. Para o escrever podemos usar um editor, por exemplo, o **emacs**;
- Agora é preciso **executar o compilador** da linguagem **C** para transformar o nosso programa numa linguagem que o processador do computador seja capaz de entender

Dizer: Olá a todos! (II)

- Acabámos de construir o programa para dizer '**Olá a todos!**'. Que fazemos com ele?
- Este programa tem de ser escrito num ficheiro que deverá ter a extensão '**.c**'. Chamemos-lhe, por exemplo, **Prog01_01.c**. Para o escrever podemos usar um editor, por exemplo, o **emacs**;
- Agora é preciso **executar o compilador** da linguagem **C** para transformar o nosso programa numa linguagem que o processador do computador seja capaz de entender, isto é, **compilar** o programa;

Dizer: Olá a todos! (II)

- Acabámos de construir o programa para dizer '**Olá a todos!**'. Que fazemos com ele?
- Este programa tem de ser escrito num ficheiro que deverá ter a extensão '**.c**'. Chamemos-lhe, por exemplo, **Prog01_01.c**. Para o escrever podemos usar um editor, por exemplo, o **emacs**;
- Agora é preciso **executar o compilador** da linguagem **C** para transformar o nosso programa numa linguagem que o processador do computador seja capaz de entender, isto é, **compilar** o programa;
- Em seguida temos de ligar o nosso **programa compilado** com outras funções já previamente compiladas (por exemplo, a definição da função **printf**).

Dizer: Olá a todos! (II)

- Acabámos de construir o programa para dizer '**Olá a todos!**'. Que fazemos com ele?
- Este programa tem de ser escrito num ficheiro que deverá ter a extensão '**.c**'. Chamemos-lhe, por exemplo, **Prog01_01.c**.
  Para o escrever podemos usar um editor, por exemplo, o **emacs**;
- Agora é preciso **executar o compilador** da linguagem **C** para transformar o nosso programa numa linguagem que o processador do computador seja capaz de entender, isto é, **compilar** o programa;
- Em seguida temos de ligar o nosso **programa compilado** com outras funções já previamente compiladas (por exemplo, a definição da função **printf**).
  Esta segunda acção designa-se por **link**.

Dizer: Olá a todos! (III)

- Para programas simples estas duas etapes podem fazer-se em simultâneo através de:

  gcc -o Prog01_01 Prog01_01.c

Dizer: Olá a todos! (III)

- Para programas simples estas duas etapes podem fazer-se em simultâneo através de:

  gcc -o Prog01_01 Prog01_01.c

  este comando, dado ao compilador de **C** (**gcc**), diz que queremos *compilar* e *linkar* o programa **Prog01_01.c**

Dizer: Olá a todos! (III)

- Para programas simples estas duas etapes podem fazer-se em simultâneo através de:

  gcc -o Prog01_01 Prog01_01.c

  este comando, dado ao compilador de **C** (**gcc**), diz que queremos *compilar* e *linkar* o programa **Prog01_01.c** e que o **programa executável** gerado terá o nome **Prog01_01**, isto é, o argumento da opção '**-o**'.

Dizer: Olá a todos! (III)

- Para programas simples estas duas etapes podem fazer-se em simultâneo através de:

  gcc -o Prog01_01 Prog01_01.c

  este comando, dado ao compilador de **C** (**gcc**), diz que queremos *compilar* e *linkar* o programa **Prog01_01.c** e que o **programa executável** gerado terá o nome **Prog01_01**, isto é, o argumento da opção '**-o**'.
- Finalmente podemos executar o programa. Para tal, escremos o nome do programa na *shell*:

Dizer: Olá a todos! (III)

- Para programas simples estas duas etapes podem fazer-se em simultâneo através de:

  gcc -o Prog01_01 Prog01_01.c

  este comando, dado ao compilador de **C** (**gcc**), diz que queremos *compilar* e *linkar* o programa **Prog01_01.c** e que o **programa executável** gerado terá o nome **Prog01_01**, isto é, o argumento da opção '**-o**'.
- Finalmente podemos executar o programa. Para tal, escremos o nome do programa na *shell*:

  ./Prog01_01 (ou apenas: **Prog01_01**)

Dizer: Olá a todos! (III)

- Para programas simples estas duas etapes podem fazer-se em simultâneo através de:

  gcc -o Prog01_01 Prog01_01.c

  este comando, dado ao compilador de **C** (**gcc**), diz que queremos *compilar* e *linkar* o programa **Prog01_01.c** e que o **programa executável** gerado terá o nome **Prog01_01**, isto é, o argumento da opção '**-o**'.

- Finalmente podemos executar o programa. Para tal, escremos o nome do programa na *shell*:

  ./Prog01_01 (ou apenas: **Prog01_01**)
  Olá a todos!

Dizer: Olá a todos! (IV) ('Prog01_02e04.c')

- Podemos dar um pouco de melhor aspecto e informação ao programa escrevendo alguns **comentários**;

```
#include <stdio.h>
main ()
{
   printf ("Ola a todos!\n");
   retrun 0;
}
```

Dizer: Olá a todos! (IV) ('Prog01_02e04.c')

- Podemos dar um pouco de melhor aspecto e informação ao programa escrevendo alguns **comentários**;
- Os **comentários** em **C** iniciam-se por /* e terminam por */. São particularmente úteis para entender o códigos, especialmente quando os programas atingem dimensões apreciáveis.

```
#include <stdio.h>
main ()
{
   printf ("Ola a todos!\n");
   retrun 0;
}
```

Dizer: Olá a todos! (IV) ('Prog01_02e04.c')

- Podemos dar um pouco de melhor aspecto e informação ao programa escrevendo alguns **comentários**;
- Os **comentários** em **C** iniciam-se por /* e terminam por */. São particularmente úteis para entender o códigos, especialmente quando os programas atingem dimensões apreciáveis.

```
/*
    Primeiro Programa em C - Versao: 1.2
*/
#include <stdio.h>
main ()
{
  printf ("Ola a todos!\n");
  retrun 0;
}
```

Dizer: Olá a todos! (IV) ('Prog01_02e04.c')

- Podemos dar um pouco de melhor aspecto e informação ao programa escrevendo alguns **comentários**;
- Os **comentários** em **C** iniciam-se por /* e terminam por */. São particularmente úteis para entender o códigos, especialmente quando os programas atingem dimensões apreciáveis.
- Em **'Prog01_03.c'** podem ver-se pequenas variantes da função '**printf**'. Em **'Prog01_04.c'** exemplifica-se a mudança de linha.

```
/*
    Primeiro Programa em C - Versao: 1.2
*/
#include <stdio.h>
main ()
{
  printf ("Ola a todos!\n");
  retrun 0;
}
```

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.

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



}
```

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

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



}
```

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

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


   c = 2.54 * p;

}
```

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;

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


   c = 2.54 * p;

}
```

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;
- Como **c** e **p** são números reais, declaramo-los como tipo '**float**';

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



   c = 2.54 * p;


}
```

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;
- Como **c** e **p** são números reais, declaramo-los como tipo '**float**';

```
#include <stdio.h>
int main ()
{
   float p, c ;

   c = 2.54 * p;

}
```

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;
- Como **c** e **p** são números reais, declaramo-los como tipo '**float**';

```
#include <stdio.h>
int main ()
{
   float p, c ;

   c = 2.54 * p;

}
```

- Se quisermos converter 2 in em cm, atribuimos a **p** o valor **2.**;

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;
- Como **c** e **p** são números reais, declaramo-los como tipo '**float**';

```
#include <stdio.h>
int main ()
{
   float p, c ;
   p = 2.;
   c = 2.54 * p;


}
```

- Se quisermos converter 2 in em cm, atribuimos a **p** o valor **2.**;

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;
- Como **c** e **p** são números reais, declaramo-los como tipo '**float**';

```
#include <stdio.h>
int main ()
{
   float p, c ;
   p = 2.;
   c = 2.54 * p;


}
```

- Se quisermos converter 2 in em cm, atribuimos a **p** o valor **2.**;
- O resultado que queremos, vamos guardá-lo na variável **c**;

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;
- Como **c** e **p** são números reais, declaramo-los como tipo '**float**';

```
#include <stdio.h>
int main ()
{
   float p, c ;
   p = 2.;
   c = 2.54 * p;


}
```

- Se quisermos converter 2 in em cm, atribuimos a **p** o valor **2.**;
- O resultado que queremos, vamos guardá-lo na variável **c**;

Converter polegadas em centímetros (I)

- Um programa um pouco mais complicado consiste, por exemplo, em **converter polegadas em centímetros (Prog02_01.c)**.
- É sabido que uma polegada mede aproximadamente 2.54 cm:

$$c = 2.54 \times p$$

- Sempre que usamos variáveis em **C** elas têm **obrigatoriamente** de ser **declaradas** de acordo com o seu **tipo**;
- Como **c** e **p** são números reais, declaramo-los como tipo '**float**';

```
#include <stdio.h>
int main ()
{
   float p, c ;
   p = 2.;
   c = 2.54 * p;
   printf ("Res: %f\n", c);
}
```

- Se quisermos converter 2 in em cm, atribuimos a **p** o valor **2.**;
- O resultado que queremos, vamos guardá-lo na variável **c**;
- No argumento da função **printf** colocamos **%f** onde queremos que o valor de **c** seja inserido.

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):
  - **cygwin**

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):
  - **cygwin**
  - **Code::Blocks**

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):
  - **cygwin**
  - **Code::Blocks**
  - **CodeLite**

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):
  - **cygwin**
  - **Code::Blocks**
  - **CodeLite**
  - **DevC++**

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):
  - **cygwin**
  - **Code::Blocks**
  - **CodeLite**
  - **DevC++**
- O primeiro é um ambiente de trabalho tipo unix para Windows;

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):
  - **cygwin**
  - **Code::Blocks**
  - **CodeLite**
  - **DevC++**
- O primeiro é um ambiente de trabalho tipo unix para Windows;
- Os seguintes, são três programas mais ou menos do mesmo tipo, disponibizando cada um deles num ambiente integrado de desenvolvimento.

Como compilar, editar e executar, em Microsoft Windows, programas em C

- Para **criar**, **editar** e **executar** programas escritos em **C**, em sistemas **Microsoft Windows**, estão disponíveis na página da cadeira explicações para a instalação dos seguintes programas (ver **HowTo** na página da cadeira):
  - **cygwin**
  - **Code::Blocks**
  - **CodeLite**
  - **DevC++**
- O primeiro é um ambiente de trabalho tipo unix para Windows;
- Os seguintes, são três programas mais ou menos do mesmo tipo, disponibizando cada um deles num ambiente integrado de desenvolvimento.
- Qualquer destes programas tem licença de software livre.

Como simular um ambiente unix em Microsoft Windows - Cygwin

- O **Cygwin** é um ambiente de tipo **Unix** para **Microsoft Windows**.

Como simular um ambiente unix em Microsoft Windows - Cygwin

- O **Cygwin** é um ambiente de tipo **Unix** para **Microsoft Windows**.
- Disponibiliza um **terminal** com a respectiva "**shell**" bem como uma **interface de janelas** (**X11**).

Como simular um ambiente unix em Microsoft Windows - Cygwin

- O **Cygwin** é um ambiente de tipo **Unix** para **Microsoft Windows**.
- Disponibiliza um **terminal** com a respectiva "**shell**" bem como uma **interface de janelas** (**X11**).
- A sua instalação apresenta uma vasta gama de pacotes de software que habitualmente são encontrados em sistemas **Unix**.

Como simular um ambiente unix em Microsoft Windows - Cygwin

- O **Cygwin** é um ambiente de tipo **Unix** para **Microsoft Windows**.
- Disponibiliza um **terminal** com a respectiva "**shell**" bem como uma **interface de janelas** (**X11**).
- A sua instalação apresenta uma vasta gama de pacotes de software que habitualmente são encontrados em sistemas **Unix**.
- Para quem não disponha de um sistema **Unix**, o **Cygwin** disponibiliza um ambiente de trabalho próximo do que se pode encontrar nos computadores das aulas laboratoriais.

Como simular um ambiente unix em Microsoft Windows - Cygwin

- O **Cygwin** é um ambiente de tipo **Unix** para **Microsoft Windows**.
- Disponibiliza um **terminal** com a respectiva "**shell**" bem como uma **interface de janelas** (**X11**).
- A sua instalação apresenta uma vasta gama de pacotes de software que habitualmente são encontrados em sistemas **Unix**.
- Para quem não disponha de um sistema **Unix**, o **Cygwin** disponibiliza um ambiente de trabalho próximo do que se pode encontrar nos computadores das aulas laboratoriais.
- A sua instalação é simples e encontra-se explicada na secção "**HowTo**" da página da cadeira de Programação.