13ª Aula - Instruções Condicionais. Ciclos. Pré-processador. Variáveis de ambiente.

Programação
Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Instruções Condicionais. ('Prog23_01.c')

- **Instruções Condicionais** são instruções que permitem **decidir**, num dado ponto dum programa, quais as instruções a executar, ou seja, qual **o caminho a seguir**.

Instruções Condicionais. ('Prog23_01.c')

- **Instruções Condicionais** são instruções que permitem **decidir**, num dado ponto dum programa, quais as instruções a executar, ou seja, qual **o caminho a seguir**.
- São instruções condicionais:

Instruções Condicionais. ('Prog23_01.c')

- **Instruções Condicionais** são instruções que permitem **decidir**, num dado ponto dum programa, quais as instruções a executar, ou seja, qual **o caminho a seguir**.
- São instruções condicionais:
  - '**if**' e '**if ... else**';
  - '**switch**'

Instruções Condicionais. ('Prog23_01.c')

- **Instruções Condicionais** são instruções que permitem **decidir**, num dado ponto dum programa, quais as instruções a executar, ou seja, qual **o caminho a seguir**.
- São instruções condicionais:
  - '**if**' e '**if ... else**';
  - '**switch**'
- Existe ainda um operador, que é habitualmente usado para **atribuições de valores**, e que permite decidir qual o **valor a atribuir** a variável. Esse operador é:

Instruções Condicionais. ('Prog23_01.c')

- **Instruções Condicionais** são instruções que permitem **decidir**, num dado ponto dum programa, quais as instruções a executar, ou seja, qual **o caminho a seguir**.
- São instruções condicionais:
  - '**if**' e '**if ... else**';
  - '**switch**'
- Existe ainda um operador, que é habitualmente usado para **atribuições de valores**, e que permite decidir qual o **valor a atribuir** a variável. Esse operador é:

  (**Condição ? Se_Verdadeiro : Se_Falso**);

  Por exemplo:

  a1 = (**x > 0** ? **x** : **-x**);

  é equivalente a:

  if (**x > 0**) a1 = **x**; **else** a1 = **(–x);**

Instruções Condicionais 'if' e 'if ... else' ('Prog23_02.c')

- A sintaxe da instrução '**if ... else**' é:

```
if (Condição)
  { Bloco_se_Verdadeiro }
else
  { Bloco_se_Falso }
```

Instruções Condicionais 'if' e 'if ... else' ('Prog23_02.c')

- A sintaxe da instrução '**if ... else**' é:

  if (**Condição**)

  { **Bloco_se_Verdadeiro** }

  else

  { **Bloco_se_Falso** }

- Note-se que o '**else**' e o '**Bloco_se_Falso**' podem ser omitidos. Ficando simplesmente a condição '**if**' e o respectivo bloco.

Instruções Condicionais 'if' e 'if ... else' ('Prog23_02.c')

- A sintaxe da instrução '**if ... else**' é:

  if (**Condição**)
  { **Bloco_se_Verdadeiro** }
  else
  { **Bloco_se_Falso** }

- Note-se que o '**else**' e o '**Bloco_se_Falso**' podem ser omitidos. Ficando simplesmente a condição '**if**' e o respectivo bloco.
- É válido escrever instruções '**if**' no interior umas das outras.

Instruções Condicionais 'if' e 'if ... else' ('Prog23_02.c')

- A sintaxe da instrução '**if ... else**' é:

  if (**Condição**)
  { **Bloco_se_Verdadeiro** }
  else
  { **Bloco_se_Falso** }

- Note-se que o '**else**' e o '**Bloco_se_Falso**' podem ser omitidos. Ficando simplesmente a condição '**if**' e o respectivo bloco.
- É válido escrever instruções '**if**' no interior umas das outras.
- Quando se têm situações com **condições sucessivas** é, por vezes, cómodo escrever essas condições na forma:

  '**if**' {...} '**else if**' {...} '**else if**' {...} '**else**' {...}

Instrução Condicional 'switch' ('Prog23_03.c')

- Quando se têm condições sucessivas controladas por um **inteiro** (por vezes, na forma de um enumerado, '**enum**'), é mais simples (e mais conveniente) usar a instrução '**switch**'.

Instrução Condicional 'switch' ('Prog23_03.c')

- Quando se têm condições sucessivas controladas por um **inteiro** (por vezes, na forma de um enumerado, '**enum**'), é mais simples (e mais conveniente) usar a instrução '**switch**'.
- A sua sintaxe é a seguinte:

```
switch (Variável_Inteira)
  {
   case i_1 :
        bloco_1
     ...
   case i_n :
        bloco_n
   default :
        bloco_se_tudo_for_falso
  }
```

Instrução Condicional 'switch' ('Prog23_03.c')

- Quando se têm condições sucessivas controladas por um **inteiro** (por vezes, na forma de um enumerado, '**enum**'), é mais simples (e mais conveniente) usar a instrução '**switch**'.
- A sua sintaxe é a seguinte:

```
switch (Variável_Inteira)
  {
   case i_1 :
        bloco_1
     ...
   case i_n :
        bloco_n
   default :
        bloco_se_tudo_for_falso
  }
```

- A interrupção da execução das instruções de um **switch** é feita com a instrução **break;**

Ciclos ('Prog24_01e2.c')

- Existem três instruções de **ciclos** em **C**:

Ciclos ('Prog24_01e2.c')

- Existem três instruções de **ciclos** em **C**:
  - '**while**' – **testa** a condição e **executa** o bloco:

    while (**Condição**) {**bloco_de_instruções**}

Ciclos ('Prog24_01e2.c')

- Existem três instruções de **ciclos** em **C**:
  - '**while**' – **testa** a condição e **executa** o bloco:

    while (**Condição**) {**bloco_de_instruções**}
  - '**do .. while**' – **executa** o bloco e **testa** a condição:

    do {**bloco_de_instruções**} **while** (**Condição**);

Ciclos ('Prog24_01e2.c')

- Existem três instruções de **ciclos** em **C**:
  - '**while**' – **testa** a condição e **executa** o bloco:

    while (**Condição**) {**bloco_de_instruções**}
  - '**do .. while**' – **executa** o bloco e **testa** a condição:

    do {**bloco_de_instruções**} **while** (**Condição**);
  - '**for**' – **inicializa** o loop, **testa** a condição e **executa** o bloco:

    for (**Inicializa** ; **Condição** ; **Incremento**) {**bloco_instruções**}

Ciclos ('Prog24_01e2.c')

- Existem três instruções de **ciclos** em **C**:
  - '**while**' – **testa** a condição e **executa** o bloco:

    while (**Condição**) {**bloco_de_instruções**}
  - '**do .. while**' – **executa** o bloco e **testa** a condição:

    do {**bloco_de_instruções**} **while** (**Condição**);
  - '**for**' – **inicializa** o loop, **testa** a condição e **executa** o bloco:

    for (**Inicializa** ; **Condição** ; **Incremento**) {**bloco_instruções**}
- Um **ciclo** pode ser **terminado** de duas maneiras: por a **condição** deixar de ser satisfeita ou **forçando a saída** com '**break;**'.

Ciclos ('Prog24_01e2.c')

- Existem três instruções de **ciclos** em **C**:
  - '**while**' – **testa** a condição e **executa** o bloco:

    while (**Condição**) {**bloco_de_instruções**}
  - '**do .. while**' – **executa** o bloco e **testa** a condição:

    do {**bloco_de_instruções**} **while** (**Condição**);
  - '**for**' – **inicializa** o loop, **testa** a condição e **executa** o bloco:

    for (**Inicializa** ; **Condição** ; **Incremento**) {**bloco_instruções**}
- Um **ciclo** pode ser **terminado** de duas maneiras: por a **condição** deixar de ser satisfeita ou **forçando a saída** com '**break;**'.
- Quando se pretende sair através de um '**break;**', usa-se, por vezes, um **loop infinito** (isto é, em que a condição é sempre satisfeita), por exemplo: '**while** (1) {...}'.

Ciclos ('Prog24_01e2.c')

- Existem três instruções de **ciclos** em **C**:
  - '**while**' – **testa** a condição e **executa** o bloco:

    while (**Condição**) {**bloco_de_instruções**}
  - '**do .. while**' – **executa** o bloco e **testa** a condição:

    do {**bloco_de_instruções**} **while** (**Condição**);
  - '**for**' – **inicializa** o loop, **testa** a condição e **executa** o bloco:

    for (**Inicializa** ; **Condição** ; **Incremento**) {**bloco_instruções**}
- Um **ciclo** pode ser **terminado** de duas maneiras: por a **condição** deixar de ser satisfeita ou **forçando a saída** com '**break;**'.
- Quando se pretende sair através de um '**break;**', usa-se, por vezes, um **loop infinito** (isto é, em que a condição é sempre satisfeita), por exemplo: '**while** (1) {...}'.
- Se se pretende passar à **iteração seguinte** ignorando as restantes instruções do bloco, usa-se a instrução '**continue;**'.

'goto' e 'label' ('Prog24_01a3.c')

- Muito embora a instrução '**goto**' não tenha directamente a ver com ciclos, foi em diversas linguagens de programação o **único modo de os implementar**.

'goto' e 'label' ('Prog24_01a3.c')

- Muito embora a instrução '**goto**' não tenha directamente a ver com ciclos, foi em diversas linguagens de programação o **único modo de os implementar**.
- O seu uso gera sempre **grandes polémicas** e muitos são os fundamentalistas que, por certo, **juraram nunca o usar**...

'goto' e 'label' ('Prog24_01a3.c')

- Muito embora a instrução '**goto**' não tenha directamente a ver com ciclos, foi em diversas linguagens de programação o **único modo de os implementar**.
- O seu uso gera sempre **grandes polémicas** e muitos são os fundamentalistas que, por certo, **juraram nunca o usar**...
- Sem sermos tão radicais é bom usá-lo **muito moderamente**...

'goto' e 'label' ('Prog24_01a3.c')

- Muito embora a instrução '**goto**' não tenha directamente a ver com ciclos, foi em diversas linguagens de programação o **único modo de os implementar**.
- O seu uso gera sempre **grandes polémicas** e muitos são os fundamentalistas que, por certo, **juraram nunca o usar**...
- Sem sermos tão radicais é bom usá-lo **muito moderamente**...
- O seu uso é feito sempre dentro de funções, e exige a existência de uma **marca** (**label**) para onde é direccionado o fluxo do programa (o label pode estar **antes** ou **depois** do **goto**):

```
goto marca;
    código
marca:
    código
```

'goto' e 'label' ('Prog24_01a3.c')

- Muito embora a instrução '**goto**' não tenha directamente a ver com ciclos, foi em diversas linguagens de programação o **único modo de os implementar**.
- O seu uso gera sempre **grandes polémicas** e muitos são os fundamentalistas que, por certo, **juraram nunca o usar**...
- Sem sermos tão radicais é bom usá-lo **muito moderamente**...
- O seu uso é feito sempre dentro de funções, e exige a existência de uma **marca** (**label**) para onde é direccionado o fluxo do programa (o label pode estar **antes** ou **depois** do **goto**):

```
goto marca;
    código
marca:
    código
```

- O seu uso mais comum (e comedido) é em **situações de erro** em que se pretende **terminar a execução** de uma função mas antes necessitamos de **organizar a saída**.

Pré-processador

- O **pré-processador** é, em primeira aproximação, uma espécie um editor de texto que permite aumentar o poder e as possibilidades do **C**.

Pré-processador

- O **pré-processador** é, em primeira aproximação, uma espécie um editor de texto que permite aumentar o poder e as possibilidades do **C**.
- As instruções dirigidas ao **pré-processador** chamam-se **directivas de pré-processador** e iniciam-se por pelo símbolo '#' no início da linha.

Pré-processador

- O **pré-processador** é, em primeira aproximação, uma espécie um editor de texto que permite aumentar o poder e as possibilidades do **C**.
- As instruções dirigidas ao **pré-processador** chamam-se **directivas de pré-processador** e iniciam-se por pelo símbolo '#' no início da linha.
- Iremos aqui chamar a atenção apenas para os aspectos principais.

Pré-processador

- O **pré-processador** é, em primeira aproximação, uma espécie um editor de texto que permite aumentar o poder e as possibilidades do **C**.
- As instruções dirigidas ao **pré-processador** chamam-se **directivas de pré-processador** e iniciam-se por pelo símbolo '**#**' no início da linha.
- Iremos aqui chamar a atenção apenas para os aspectos principais.
- O **pré-processamento** é feito **linha a linha** e essas linhas **não terminam** em **ponto e vírgula**. O processamento das directivas é feito por elementos básicos ('**tokens**').

Pré-processador

- O **pré-processador** é, em primeira aproximação, uma espécie um editor de texto que permite aumentar o poder e as possibilidades do **C**.
- As instruções dirigidas ao **pré-processador** chamam-se **directivas de pré-processador** e iniciam-se por pelo símbolo '**#**' no início da linha.
- Iremos aqui chamar a atenção apenas para os aspectos principais.
- O **pré-processamento** é feito **linha a linha** e essas linhas **não terminam** em **ponto e vírgula**. O processamento das directivas é feito por elementos básicos ('**tokens**').
- Como exemplo de **directivas de pré-processador** pode ver-se o ficheiro '**/usr/include/math.h**'.

Pré-processador

- O **pré-processador** é, em primeira aproximação, uma espécie um editor de texto que permite aumentar o poder e as possibilidades do **C**.
- As instruções dirigidas ao **pré-processador** chamam-se **directivas de pré-processador** e iniciam-se por pelo símbolo '**#**' no início da linha.
- Iremos aqui chamar a atenção apenas para os aspectos principais.
- O **pré-processamento** é feito **linha a linha** e essas linhas **não terminam** em **ponto e vírgula**. O processamento das directivas é feito por elementos básicos ('**tokens**').
- Como exemplo de **directivas de pré-processador** pode ver-se o ficheiro '**/usr/include/math.h**'.
- No quadro seguinte podem ver-se as **directivas de pré-processador**:

Pré-processador - Directivas

Directiva	Acção
#	Directiva nula
#include	Inclui o conteúdo de um ficheiro
#define	Define uma macro
#undef	Retira a definição de uma macro
#if	Compilação condicional
#ifdef	Compilação condicional (se existir macro)
#ifndef	Compilação condicional (se não existir macro)
#elif	Compilação condicional (corresponde a 'else if')
#else	Compilação condicional (corresponde a 'else')
#endif	Compilação condicional (fim de um bloco '#if')
#line	Controle de erro
#error	Força mensagem de erro
#pragma	Introdução de controles dependentes da implementação

Pré-processador - Define ('Prog26_01a03.c')

- Os programas **'Prog26_01.c'** e **'Prog26_02.c'** são duas representações diferentes da função quadrado de um número. Na primeira é escrita uma **função** para calcular o quadrado de um **double**, enquanto na segunda o quadrado é calculado por uma **macro**.

Pré-processador - Define ('Prog26_01a03.c')

- Os programas **'Prog26_01.c'** e **'Prog26_02.c'** são duas representações diferentes da função quadrado de um número. Na primeira é escrita uma **função** para calcular o quadrado de um **double**, enquanto na segunda o quadrado é calculado por uma **macro**.
- Como se pode ver a **macro**, uma vez que faz uma simples substituição **não necessita** da declaração do tipo pelo que pode ser usada **independentemente do tipo**.

Pré-processador - Define ('Prog26_01a03.c')

- Os programas **'Prog26_01.c'** e **'Prog26_02.c'** são duas representações diferentes da função quadrado de um número. Na primeira é escrita uma **função** para calcular o quadrado de um **double**, enquanto na segunda o quadrado é calculado por uma **macro**.
- Como se pode ver a **macro**, uma vez que faz uma simples substituição **não necessita** da declaração do tipo pelo que pode ser usada **independentemente do tipo**.
- Como se pode ver pelas macros **QUADRADO** e **QUAD** é necessário um **certo cuidado** a escrevê-las: '**2+3*2+3**' é obviamente diferente de '**(2+3)*(2+3)**'.

Pré-processador - Condições ('Prog26_04.c')

- Uma vez que as instruções ao **pré-compilador** são executadas **antes** da compilação, é possível incluir nelas diferentes **opções de código** que têm a ver com o **sistema operativo** ou outros tipos de decisão, por nós consideradas como convenientes.

Pré-processador - Condições ('Prog26_04.c')

- Uma vez que as instruções ao **pré-compilador** são executadas **antes** da compilação, é possível incluir nelas diferentes **opções de código** que têm a ver com o **sistema operativo** ou outros tipos de decisão, por nós consideradas como convenientes.
- Os ficheiros "**.h**" usam habitualmente uma **funcionalidade do pré-compilador** que é a definição de uma **variável de controle** que irá garantir que um ficheiro "**.h**" não é incluído **duas vezes** num programa:

Pré-processador - Condições ('Prog26_04.c')

- Uma vez que as instruções ao **pré-compilador** são executadas **antes** da compilação, é possível incluir nelas diferentes **opções de código** que têm a ver com o **sistema operativo** ou outros tipos de decisão, por nós consideradas como convenientes.
- Os ficheiros "**.h**" usam habitualmente uma **funcionalidade do pré-compilador** que é a definição de uma **variável de controle** que irá garantir que um ficheiro "**.h**" não é incluído **duas vezes** num programa:
- **Todo o ficheiro** está incluído dentro de um '**if**':

Pré-processador - Condições ('Prog26_04.c')

- Uma vez que as instruções ao **pré-compilador** são executadas **antes** da compilação, é possível incluir nelas diferentes **opções de código** que têm a ver com o **sistema operativo** ou outros tipos de decisão, por nós consideradas como convenientes.
- Os ficheiros "**.h**" usam habitualmente uma **funcionalidade do pré-compilador** que é a definição de uma **variável de controle** que irá garantir que um ficheiro "**.h**" não é incluído **duas vezes** num programa:
- **Todo o ficheiro** está incluído dentro de um '**if**':

```
#ifndef __NOME_H__
#define __NOME_H__ 1
...
#endif
```

Pré-processador - Condições ('Prog26_04.c')

- Uma vez que as instruções ao **pré-compilador** são executadas **antes** da compilação, é possível incluir nelas diferentes **opções de código** que têm a ver com o **sistema operativo** ou outros tipos de decisão, por nós consideradas como convenientes.
- Os ficheiros "**.h**" usam habitualmente uma **funcionalidade do pré-compilador** que é a definição de uma **variável de controle** que irá garantir que um ficheiro "**.h**" não é incluído **duas vezes** num programa:
- **Todo o ficheiro** está incluído dentro de um '**if**':

```
#ifndef __NOME_H__
#define __NOME_H__ 1
...
#endif
```

em que '**#ifndef**' significa '**se não definido**'. O '**1**' a seguir à definição não é necessário. Usualmente, para evitar confusões nas designações usadas, '**NOME**' é o próprio nome do ficheiro.

Variáveis do Ambiente ('Prog25_01a04.c')

- Existe um conjunto de **variáveis** que se encontram definidas no ambiente de trabalho sempre que fazemos um '**login**'. Essas variáveis dizer respeito à máquina em que estamos e ao nosso '**login**' específico.

Variáveis do Ambiente ('Prog25_01a04.c')

- Existe um conjunto de **variáveis** que se encontram definidas no ambiente de trabalho sempre que fazemos um '**login**'. Essas variáveis dizer respeito à máquina em que estamos e ao nosso '**login**' específico.
- Em '**unix**' podemos obtê-las com o comando '**env**' na **shell**.

Variáveis do Ambiente ('Prog25_01a04.c')

- Existe um conjunto de **variáveis** que se encontram definidas no ambiente de trabalho sempre que fazemos um '**login**'. Essas variáveis dizer respeito à máquina em que estamos e ao nosso '**login**' específico.
- Em '**unix**' podemos obtê-las com o comando '**env**' na **shell**.
- Já vimos que os dois primeiros argumentos da função '**main**' são os **argumentos do programa** quando é executado. As **variáveis de ambiente** passam para um programa através de o terceiro argumento da função '**main**': '**char** ****env**'.

Variáveis do Ambiente ('Prog25_01a04.c')

- Existe um conjunto de **variáveis** que se encontram definidas no ambiente de trabalho sempre que fazemos um '**login**'. Essas variáveis dizer respeito à máquina em que estamos e ao nosso '**login**' específico.
- Em '**unix**' podemos obtê-las com o comando '**env**' na **shell**.
- Já vimos que os dois primeiros argumentos da função '**main**' são os **argumentos do programa** quando é executado. As **variáveis de ambiente** passam para um programa através de o terceiro argumento da função '**main**': '**char** ****env**'.
- Este argumento '**env**' é, à semelhança de '**argv**', um vector de ponteiros para '**char**', no entanto, não tem um contador associado e termina com o último ponteiro em '**NULL**'.

Variáveis do Ambiente ('Prog25_01a04.c')

- Existe um conjunto de **variáveis** que se encontram definidas no ambiente de trabalho sempre que fazemos um '**login**'. Essas variáveis dizer respeito à máquina em que estamos e ao nosso '**login**' específico.
- Em '**unix**' podemos obtê-las com o comando '**env**' na **shell**.
- Já vimos que os dois primeiros argumentos da função '**main**' são os **argumentos do programa** quando é executado. As **variáveis de ambiente** passam para um programa através de o terceiro argumento da função '**main**': '**char **env**'.
- Este argumento '**env**' é, à semelhança de '**argv**', um vector de ponteiros para '**char**', no entanto, não tem um contador associado e termina com o último ponteiro em '**NULL**'.
- Se executarmos a file de comandos '**Prog25_04_comandos**' com um **argumento numérico**, podemos ver qual o efeito do valor do '**return status;**' da função '**main**' no **ambiente da shell**.