

Métodos Computacionais em Física

Noções Básicas de Linguagem C

Edivaldo M. Santos e João R. T. de Mello Neto

Aula 2

- ▶ Diretivas de préprocessamento
- ▶ Identificadores
- ▶ Tipos
- ▶ Variáveis e Aritmética
- ▶ Operadores e precedência
- ▶ Efeitos colaterais
- ▶ Laços: if, while and do, for, switch, break, continue
- ▶ Funções
- ▶ Passagem de argumentos
- ▶ Arrays
- ▶ Entrada e Saída
- ▶ Apontadores
- ▶ Alocação Dinâmica de Memória
- ▶ Bibliotecas e Funções Matemáticas

Na aula de hoje:

- ▶ Estrutura de um programa em C
- ▶ Diretivas de pré-processamento
- ▶ instruções de entrada e saída: **printf** e **scanf**
- ▶ instrução de laço **while**
- ▶ variáveis e expressões aritméticas
- ▶ operadores binários

Começemos com um programa exemplo muito simples, denominado por programadores de **programa "Hello, world!"**, mas de extremo valor didático para os iniciantes em qualquer linguagem de programação. O programa abaixo é o conteúdo de um arquivo em formato texto denominado hello.c:

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
  
}
```

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
}
```

- Linhas de comentários num programa em C são delimitadas por `/* */`.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
  
}
```

- ▶ Linhas de comentários num programa em C são delimitadas por `/* */`.
- ▶ Essas linhas são ignoradas pelo compilador. Sua função é passar explicações a um humano e não a uma máquina.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
  
}
```

- ▶ Linhas de comentários num programa em C são delimitadas por `/* */`.
- ▶ Essas linhas são ignoradas pelo compilador. Sua função é passar explicações a um humano e não a uma máquina.
- ▶ A utilização de comentários é uma prática essencial à tarefa de programação. Programas sem comentários, a menos que extremamente simplificados, tornam-se praticamente ininteligíveis aos usuários que não participaram de sua programação e até mesmo para o autor do código.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

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

- O símbolo # indica a presença de uma **diretiva de pré-processamento**. Nesse caso específico, uma diretiva do tipo #include.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

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

- ▶ O símbolo # indica a presença de uma **diretiva de pré-processamento**. Nesse caso específico, uma diretiva do tipo #include.
- ▶ Como o próprio nome sugere, o pré-processamento é uma etapa anterior à compilação. A diretiva em questão é uma instrução ao pré-processor de que há algo a ser feito antes de entregar o código ao compilador. No caso da diretiva #include, a instrução é de que o *header file* correspondente à biblioteca stdio deve ser incluída.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

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

- ▶ Essa linha inicia a definição da **função** chamada principal (ou main).
Todo programa em C deve conter uma função chamada main.
Quando executa-se um programa em C, o que estamos fazendo é executar as instruções contidas na função principal.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

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

- ▶ Essa linha inicia a definição da **função** chamada principal (ou main).
Todo programa em C deve conter uma função chamada main.
Quando executa-se um programa em C, o que estamos fazendo é executar as instruções contidas na função principal.
- ▶ Toda função em C deve possuir um tipo. A palavra int logo antes de main indica que a função principal do nosso programa hello.c é do tipo inteiro. **Em linguagem C, a função main deve necessariamente ser do tipo int.**

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("hello, world!");
```

```
    return 0;
```

```
}
```

- ▶ Ao declarar-se a função main como inteira (int), estamos assumindo o compromisso de que essa função deve retornar a quem a invocou, um número inteiro.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

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

- ▶ Ao declarar-se a função main como inteira (int), estamos assumindo o compromisso de que essa função deve retornar a quem a invocou, um número inteiro.
- ▶ Os paranteses logo após a palavra main englobam os chamados parâmetros da função. Nesse caso específico, nenhum parâmetro é passado.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("hello, world!");
```

```
    return 0;
```

```
}
```

- ▶ As chaves delimitam o escopo da função main e de qualquer função em C. Elas definem o limite do corpo da função. Variáveis criadas no interior dessas chaves só existem dentro do escopo da sua respectiva função.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("hello, world!");
```

```
    return 0;
```

```
}
```

- ▶ As chaves delimitam o escopo da função main e de qualquer função em C. Elas definem o limite do corpo da função. Variáveis criadas no interior dessas chaves só existem dentro do escopo da sua respectiva função.
- ▶ No interior das chaves são incluídas as **instruções** (ou *statements* em inglês). As chaves informam ao compilador de C para tratar todas as instruções no seu interior como uma unidade.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
  
}
```

- ▶ A primeira instrução no escopo de nossa função main pertence à classe de instruções do tipo **entrada e saída**. Ao ser executada, a expressão "hello, world!" deve ser impressa no dispositivo padrão de saída, que na maioria das implementações é o terminal do seu computador.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
  
}
```

- ▶ A primeira instrução no escopo de nossa função main pertence à classe de instruções do tipo **entrada e saída**. Ao ser executada, a expressão "hello, world!" deve ser impressa no dispositivo padrão de saída, que na maioria das implementações é o terminal do seu computador.
- ▶ E encontramos nosso segundo exemplo de função em C: printf. Ao contrário de nossa função main que não recebe qualquer parâmetro, printf recebe um conjunto de caracteres e imprime-os na tela. Você deve estar se perguntando sobre o tipo da função printf. Assim como main, printf em C é do tipo int. Instruções de entrada e saída formatadas serão estudadas em mais detalhes em breve.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
  
}
```

- E finalmente temos a instrução de retorno identificada pela **palavra-chave** `return`. Como prometido na declaração da função `main`, o valor retornado por ela (0) é do tipo inteiro.

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
  
int main()  
{  
  
    printf("hello, world!");  
  
    return 0;  
  
}
```

- ▶ E finalmente temos a instrução de retorno identificada pela **palavra-chave** return. Como prometido na declaração da função main, o valor retornado por ela (0) é do tipo inteiro.
- ▶ É possível compilar um código em C cuja função main, apesar de declarada do tipo inteira, não contenha uma instrução de retorno. **Mas esta é uma prática NÃO RECOMENDADA.** A instrução de retorno permite à unidade que invocou main (ou qualquer outra função) saber se todas as instruções foram executadas com sucesso, verificando o valor retornado por main.

Você deve estar se perguntando qual o verdadeiro papel no programa `hello.c` da diretiva de pré-processamento

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

já que não fizemos nenhum comentário adicional sobre ela ao analisarmos o corpo da função principal. No entanto, ela foi de grande importância para o programa, e isso tem a ver com a diferença entre [◀ Linguagem Núcleo e Bibliotecas em C](#).

Compile o programa hello.c executando o seguinte comando em um terminal do seu computador:

```
cc -Wall -pedantic -o hello hello.c
```

O arquivo hello executável deve ter sido criado no diretório atual. Execute-o com:

```
./hello
```

seguido de ENTER.

```
#include <stdio.h>

/* imprime uma tabela Fahrenheit-Celsius para temperaturas
   Fahrenheit = 0, 20, ..., 300 */
int main()
{
    int fahr, celsius;
    int inf, sup, passo;
    inf = 0;    /* limite inferior da escala de temperaturas */
    sup = 300; /* limite superior */
    passo = 20; /* tamanho do passo */
    fahr = inf;
    while (fahr <= sup) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + passo;
    }
    return 0;
}
```

```
#include <stdio.h>

/* imprime uma tabela Fahrenheit-Celsius para temperaturas
   Fahrenheit = 0, 20, ..., 300 */
int main()
{
    int fahr, celsius;
    int inf, sup, passo;
    ...
}
```

- ▶ Essas instruções fazem a **declaração** de cinco **variáveis**. Na declaração anuncia-se o tipo e o nome das variáveis. **Em C, todas as variáveis devem ser declaradas antes de serem usadas**. Aqui, declara-se as variáveis fahr, celsius, inf, sup e passo como sendo todas do tipo inteiro (int).

```
#include <stdio.h>

/* imprime uma tabela Fahrenheit-Celsius para temperaturas
   Fahrenheit = 0, 20, ..., 300 */
int main()
{
    ...
    inf = 0;    /* limite inferior da escala de temperaturas */
    sup = 300;  /* limite superior */
    passo = 20; /* tamanho do passo */
    fahr = inf;
    ...
}
```

- ▶ Inicia-se o cálculo da conversão das temperaturas, fazendo-se uma **atribuição** de valores para as variáveis previamente declaradas. A temperatura inferior é 0 °F, a superior 300 °F, a diferença de temperaturas em Fahrenheit é 20 °F. A variável fahr recebe o valor de inf, cujo valor foi previamente atribuído.

```
#include <stdio.h>

/* imprime uma tabela Fahrenheit-Celsius para temperaturas
   Fahrenheit = 0, 20, ..., 300 */
int main()
{
    ...
    while (fahr <= sup) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + passo;
    }
    ...
}
```

- ▶ Como a operação de conversão envolve sempre o mesmo algoritmo, utilizamos uma instrução de laço (*loop*) do tipo **while**. Ao encontrar uma instrução do tipo while, a condição entre parenteses é testada. Se ela é verdadeira, as instruções delimitadas pelos colchetes são executadas. Esse procedimento é repetido até que a condição se torne falsa, e nesse caso o processamento segue o fluxo normal, ou seja, as instruções seguintes ao escopo do while passam a ser executadas.

```
#include <stdio.h>

...
int main()
{
    ...
    while (fahr <= sup) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + passo;
    }
    ...
}
```

- ▶ No nosso programa de conversão há três instruções a serem executadas: duas instruções **aritméticas** e uma de **entrada/saída**. A primeira instrução faz a conversão para o valor atual de temperatura Fahrenheit, a segunda imprime na tela os valores da temperatura em Fahrenheit e seu respectivo valor na escala Celsius; a terceira instrução muda o valor de temperatura Fahrenheit para o próximo valor da tabela.

```
#include <stdio.h>

...
int main()
{
    ...
    while (fahr <= sup) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + passo;
    }
    ...
}
```

- ▶ No nosso programa de conversão há três instruções a serem executadas: duas instruções **aritméticas** e uma de **entrada/saída**. A primeira instrução faz a conversão para o valor atual de temperatura Fahrenheit, a segunda imprime na tela os valores da temperatura em Fahrenheit e seu respectivo valor na escala Celsius; a terceira instrução muda o valor de temperatura Fahrenheit para o próximo valor da tabela.
- ▶ Esse procedimento será repetido até que a variável fahr ultrapasse o valor máximo $\text{sup} = 300$.

```
...  
printf("%d\t%d\n", fahr, celsius);  
...
```

- ▶ Você certamente deve ter notado a diferença de argumentos passado á função printf nesse exemplo comparado ao programa hello.c. Dessa vez, três argumentos foram passados a printf: um conjunto de caracteres delimitados por aspas, e dois inteiros (fahr e celsius).

```
...  
printf("%d\t%d\n", fahr, celsius);  
...
```

- ▶ Você certamente deve ter notado a diferença de argumentos passado á função printf nesse exemplo comparado ao programa hello.c. Dessa vez, três argumentos foram passados a printf: um conjunto de caracteres delimitados por aspas, e dois inteiros (fahr e celsius).
- ▶ Ao receber os argumentos acima, a função printf é informada de que deve imprimir na tela dois números inteiros, alinhados com um TAB entre eles e, além disso, deve-se pular linha após a impressão. Essa informação está contida no primeiro argumento passado à função, que é basicamente uma informação de formato. %d implica que o segundo argumento é do tipo inteiro, \t produz um TAB no dispositivo de saída, o segundo %d informa que o terceiro argumento é também do tipo inteiro e, finalmente, \n ocasiona um salto de linha na tela.

- ▶ Em resumo, o formato dos argumentos de printf é

```
printf("instrucoes de formato", lista de variaveis para impressao);
```

onde variáveis a ser impressas deve estar separadas por vírgula. printf irá imprimir cada variável que aparece na lista, segundo os identificadores de formato e na ordem em que eles aparecem no primeiro argumento. Se o número de variáveis não corresponde ao número de identificadores, um erro ocorrerá já na fase de compilação.

- ▶ Em resumo, o formato dos argumentos de printf é

```
printf("instrucoes de formato", lista de variaveis para impressao);
```

onde variáveis a ser impressas deve estar separadas por vírgula. printf irá imprimir cada variável que aparece na lista, segundo os identificadores de formato e na ordem em que eles aparecem no primeiro argumento. Se o número de variáveis não corresponde ao número de identificadores, um erro ocorrerá já na fase de compilação.

- ▶ Outros identificadores de formato que podem ser usados com printf são:

```
%d    imprime um inteiro decimal
%6d    imprime um inteiro decimal com pelo menos 6 digitos
%f    imprime um ponto flutuante
%6f    imprime um ponto flutuante com pelo menos 6 digitos
%.2f    imprime um ponto flutuante, com 2 digitos apos o ponto decimal
%6.2f    imprime um ponto flutuante, com pelo menos 6 digitos no total, dos quais 2 apos o ponto decimal
```

- ▶ Até agora só utilizamos a função `printf` para instruções de saída. E se estivermos interessados em instruções de entrada? Por exemplo, se estamos interessados em ler números digitados no teclado pelo usuário para que sirvam de dados de entrada em um programa. Observe a sequência de duas instruções:

```
...  
printf("\nDigite um n'umero inteiro: ");  
scanf("%d", &n);  
...
```

- ▶ A primeira imprime a mensagem "Digite um número inteiro:". A segunda instrução lê caracteres digitados pelo usuário utilizando o dispositivo de entrada padrão, isto é, o teclado. Assim como `printf`, `scanf` utiliza o primeiro argumento para definir os tipos das variáveis a serem lidas do terminal. O valor de retorno de `scanf` é igual ao número de entradas lidas com sucesso a partir do dispositivo de entrada. Nessa caso específico, apenas uma entrada é esperada, e espera-se que ela seja do tipo inteiro.
- ▶ A execução da instrução `scanf`, ocasiona uma parada na sequência de execuções que só é retomada quando o usuário pressiona a tecla ENTER, de modo que o **buffer** de entrada é de fato lido por `scanf`.

PARA FAZER EM SALA DE AULA:

- Modifique o programa de conversão Fahrenheit-Celsius para que os valores de temperatura sejam números reais representados por ponto flutuante (float) em precisão simples.
 - Feito isso, faça com que seu programa imprima não somente os valores em $^{\circ}\text{F}$ e $^{\circ}\text{C}$, mas também o valor de temperatura na escala absoluta Kelvin.
 - Por fim, imprima também uma quarta coluna, contendo o valor $\frac{3}{2}k_B T$, que representa a energia cinética média de translação de uma molécula num gás ideal monoatômico à temperatura T , onde $k_B = 1.3806504 \times 10^{-23}$ J/K. Os valores impressos na tela devem estar em unidades de elétron-volts (eV): $1 \text{ eV} = 1.60217653 \times 10^{-19}$ J. A diretiva de pré-processamento `#define` pode ser útil aqui.
- Escreva um programa usando as funções `printf`, `scanf` e a instrução de laço **while** para que: dado um número inteiro positivo n qualquer digitado pelo usuário, imprima os n primeiros naturais ímpares.

EXERCICIO PROGRAMA 1 - EP1 - (ENTREGA: ATÉ 05/04/2010)

1. Em álgebra, um número n é dito perfeito se for igual à soma de seus divisores positivos e diferentes de n . Por exemplo, 6 é perfeito, pois $6=1+2+3$. Escreva um programa que identifique todos os números perfeitos entre 1 e 1000. Para isso, você necessitará dos **operadores binários** `%` e `==`. Em C, a instrução:

`a % b`

retorna o resto da divisão do inteiro `a` pelo inteiro `b`. Enquanto

`a == b`

retorna 1 (verdadeiro) ou 0 (falso) se "`a` igual a `b`" ou se "`a` diferente de `b`", respectivamente. A comparação também pode ser feita usando-se o operador binário `!=` ("diferente de").

Você necessitará também conhecer a instrução de execução condicional `if`. Por exemplo:

```
if (a%b == 0){  
    ...  
}
```

executa as possíveis intruções delimitadas por chaves se `a` é um múltiplo de `b`.

Como o próprio nome sugere, o pré-processamento é uma etapa anterior à compilação. A diretiva de pré-processamento é uma instrução ao pré-processor de que há algo a ser feito antes de entregar o código ao compilador.

Diretivas de pré-processamento podem englobar instruções diversas:

- ▶ As diretivas

```
#include <header_file.h>
```

ou

```
#include 'header_file.h'
```

são instruções para que um determinado *header file* seja incluído no código fonte. No primeiro caso, o arquivo corresponde a uma biblioteca padrão do C, de forma que deve ser envolvido pelos símbolos `<>`, já no segundo caso, o arquivo não faz parte das bibliotecas padrões do C e deve ser envolvido por aspas `" "`.

- ▶ A diretiva mais simples que pode ser incluída em C é a chamada diretiva nula:

```
#
```

e como o próprio nome sugere, sua inclusão não acarreta a realização de qualquer tarefa de pré-processamento.

- diretiva de definição de macros:

```
#define MACRO TEXTO_SUBSTITUTIVO
```

Muitas vezes pode ser desejável substituir um certo pedaço de código que aparece muitas vezes no arquivo fonte por uma expressão mais curta ou mais intuitiva. Por exemplo, seu programa pode utilizar o valor do número π muitas vezes. Ao invés de digitar 3.1415926535897932 todas as vezes, você pode utilizar uma diretiva `#define`:

```
#define PI 3.1415926535897932
.
.
.
/* conversao de x graus para y radianos */
    y = x*PI/180.;
```

e durante o pré-processamento, todas as ocorrências da macro `PI` serão substituídas por 3.1415926535897932.

- Vejamos outros exemplos:

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
#define HELLO printf("hello, world!")  
  
int main()  
{  
  
    /* Duas formas de dizer Alo ao mundo */  
    HELLO;  
    printf("hello, world!");  
  
    return 0;  
  
}
```

► Vejamos outros exemplos:

```
/* um pequeno programa Hello, world! */  
#include <stdio.h>  
#define H1 printf("hello, world!")  
#define H2(a) printf(a)  
  
int main()  
{  
  
    /* Tres formas de dizer Alo ao mundo */  
    H1;  
    H2("hello, world!");  
    printf("hello, world!");  
  
    return 0;  
  
}
```

- Anula a definição de uma macro

```
#undef MACRO
```

mesmo que ela não tenha sido previamente definida.

- As seguintes diretivas são usadas para compilação condicional:

```
# if  
# ifdef  
# ifndef  
# elif  
# else  
# endif
```

- Muitas vezes é conveniente passar ao processador apenas certas partes do código fonte. Por exemplo, certas partes do código podem conter instruções cuja compilação e até mesmo os efeitos são dependentes da arquitetura do processador, de modo que é necessário selecionar somente a parte condizente ao tipo de máquina em que o programa executável irá rodar. Em outras ocasiões, deseja-se apenas selecionar certas partes do código que executam algoritmos específicos.

```
#include <stdio.h>
#define CONVERT 1

/* Converte de graus para radianos ou diz Alo ao mundo */
int main()
{
    ...
#ifdef CONVERT
    /* conversao de x graus para y radianos */
    ...
#else
    printf("hello, world!\n");
#endif
}
```

- ▶ Além das diretivas apresentadas até agora, há mais três:

```
#line  
#pragma  
#error
```

utilizadas para definir os nomes intrínsecos `_LINE_` e `_FILE_`, passar instruções que sejam dependentes da implementação e relatar erros, respectivamente. Elas não serão importantes para as tarefas desenvolvidas nesse curso e, de fato, são de pouca utilidade para a maioria dos programadores. O aluno interessado pode buscar mais informações na literatura sobre tais diretivas.

Há vantagens nesse tipo de divisão em núcleo e bibliotecas:

- ▶ programas em C podem tornar-se extremamente modularizados, onde cada módulo carrega somente as bibliotecas necessárias para a execução de instruções específicas, podendo ser compilados como módulos totalmente separados de outras partes do programa completo. Quando uma determinada parte do programa necessita ser mudada, somente a parte em questão necessita ser recompilada.
- ▶ o programa carrega apenas o que necessita usar, resultando em códigos que consomem menos memória e consequentemente apresentam tempos menores de execução.