

Introdução à Programação C

Fabio Mascarenhas - 2014.2

<http://www.dcc.ufrj.br/~fabiom/introc>

Funções

- Um bom programa é *fatorado* em diversas funções pequenas, cada uma resolvendo uma parte específica do problema

```
static tipo_de_retorno nome_da_função(parâmetros) {  
    variáveis locais
```

```
    comandos
```

```
}
```

assinatura

(tipo nome, tipo nome, ...)

- Um comando essencial em uma função é o comando **return**, que encerra a execução da função atual e retorna o valor de uma expressão como o valor de retorno da função
- O corpo de uma função simples pode ser apenas um comando **return** e uma expressão envolvendo seus parâmetros

converte(100.52)

```
static double converte(double c) {  
    return 1.8 * c + 32;  
}
```

parâmetro

Parâmetros vs argumentos

- Os *parâmetros* de uma função são os nomes que damos para a sua entrada, quando escrevemos o código da função
- Em C, os parâmetros sempre são precedidos dos seus tipos
- Os *argumentos* de uma *chamada* de função são os *valores* passados para uma função, dados pelas expressões que escrevemos na chamada
- Cada argumento deve corresponder a um parâmetro, e o tipo do valor daquele argumento também deve corresponder ao tipo do parâmetro

Variáveis locais

- As variáveis declaradas no corpo de uma função são *locais* àquela função, e só existem quanto a função estiver executando
- Os parâmetros da função também são como variáveis locais
- Podemos atribuir a um parâmetro, mas isso não afeta o *argumento* que foi passado para a chamada da função!

```
static int duplica(int x) {  
    x = x * 2;  
    return x;  
}
```

```
int main()  
{  
    int x, y;  
    scanf("%d %d", &x, &y);  
    printf("%d %d\n", duplica(x), duplica(y));  
    printf("%d %d\n", x, y);  
    return 0;  
}
```

Exemplo

- Dados a hora, minuto e segundo em que um corredor de uma maratona partiu, e dados a hora, minuto e segundos em que este mesmo corredor cruzou a linha de chegada, calcule o tempo total de prova deste corredor em segundos

```
int tempo_prova(int hp, int mc, int sc, int hc, int mc, int sc)
```

- A função principal que pede para o usuário a hora, minuto e segundo de partida e de chegada do corredor, usa a função tempo_prova para calcular o tempo de prova, e imprime o tempo no formato horas:minutos:segundos

Parâmetros de saída

- Uma função C só pode retornar um único valor com `return`, mas às vezes uma função precisa retornar várias coisas
- No exemplo do slide anterior, se quisermos transformar o código que traduz segundos para horas, minutos e segundos precisamos de uma função que retorna *três* valores
- Outro exemplo é `scanf`, que retorna um número arbitrário de valores, a depender dos códigos de formato passados no modelo
- A maneira de expressar essas funções em C é com *parâmetros de saída*

Parâmetros de saída (2)

- Indicamos um parâmetro de saída adicionando * (uma estrela) ao seu tipo: `char*`, `int*`, `double*`
- Não usamos `return` para indicar parâmetros de saída, mas fazemos uma atribuição a cada um deles
- Nessa atribuição, colocamos outra estrela à esquerda do nome do parâmetro:

```
static void divr(int a, int b, int* q, int* r) {  
    *q = a / b;  
    *r = a % b;  
}
```

- Não é prudente misturar `return` e parâmetros de saída, por isso usamos o tipo de retorno `void`

Exemplo

- No exemplo anterior, vamos extrair o código que transforma segundos em horas, minutos e segundos da função principal para uma função auxiliar usando três parâmetros de saída
- Agora vamos *refatorar* esse código para usar a função `divr` dada no slide anterior

Condicionais

- Até agora todas os comandos de nossas funções são executados um após o outro, mas isso é muito limitante
- Vocês se lembram que é muito comum ter funções em que o *fluxo de controle* dentro da função depende de certas *condições*
- O comando condicional básico em Python era o comando if-elif-else, e temos um comando similar em C
- Uma diferença fundamental é que C não possui um tipo específico para valores verdadeiros e falsos: qualquer número serve, sendo 0 falso e diferente de 0 verdadeiro!

if e if-else

- O comando `if` executa um bloco de comandos apenas se a condição for verdadeira:

```
if(a > b) {  
    return a;  
}
```

⌋

- Se a condição for falsa a execução segue no comando após o `if`
- O comando `if-else` recebe dois blocos de comandos, executando *apenas* o primeiro se a condição for verdadeira e *apenas* o segundo se for falsa

```
if(a > b) {  
    return a;  
} else {  
    return b;  
}
```

Cascata de ifs

- Um comando condicional pode precisar ter mais de dois casos, usando mais de uma condição *em cascata*
- Entre o bloco `if` e o bloco `else` podemos ter quantos blocos `else if` quisermos, cada um com uma condição

```
if(x > 3) {  
    return x;  
} else if(x < -1) {  
    return -x;  
} else {  
    return 1;  
}
```

- Apenas um dos blocos é executado: ou o que corresponde à primeira condição verdadeira, ou o `else` se todas forem falsas

Expressões condicionais

- Em uma expressão condicional normalmente usamos os *operadores relacionais* (>, <, >=, <=, !=, ==), que têm como resultado 0 se a comparação for falsa e 1 se for verdadeira
- Podemos usar também os *operadores lógicos*, para combinar várias expressões condicionais:
 - && (e), que é verdadeiro se o seu lado esquerdo e direito forem verdadeiros
 $(x > 3) \ \&\& \ (x < 10)$
 - || (ou), que é verdadeiro se seu lado esquerdo ou direito forem verdadeiros
 $(x > 3) \ || \ (x < 0)$
 - ! (não), que é verdadeiro se sua expressão for falsa e vice-versa
 $! (x > 3) \equiv x <= 3$

Tabelas verdade

- Podemos esquematizar o efeito dos operadores lógicos com *tabelas verdade*

C1	C2	∧
V	V	V
V	F	F
F	V	F
F	F	F

C1	C2	∨
V	V	V
V	F	V
F	V	V
F	F	F

C	!
V	F
F	V

Predicados

- Podemos também usar chamadas de função com um resultado booleano em uma expressão condicional
- Essas funções são chamadas de *predicados*, e têm tipo de retorno `int`

```
static int letra(char c) {  
    return ((c >= 'a') && (c <= 'z')) ||  
           ((c >= 'A') && (c <= 'Z'));  
}
```

Exemplo

- Vamos criar um predicado para dizer se um aluno foi aprovado ou não, dadas as notas em suas três provas e as regras de aprovação do nosso curso