Aula 07:

Funções I

Prof. Edvard

edvard@unifei.edu.br

Universidade Federal de Itajubá

Funções

- Em programação (como em nossas próprias vidas), nos deparamos com problemas dos mais **variados níveis de dificuldade**:
  - Em nossas aulas anteriores, já estudamos desde problemas simples de física como o cálculo de **velocidade média** até problemas supercomplexos de processamento de texto, como a **indexação de todo o conteúdo da internet** realizada por sites de busca.
- Como **resolvemos** cada um deles?
  - É preciso pensar antes de agir.
  - O segredo é o planejamento.

Funções

- Problemas complexos exigem soluções mais trabalhadas, e tentar encontrá-las pensando num problema como um todo é a pior das estratégias.
  - Muitas vezes, a quantidade de informação envolvida é muito grande e, sem planejamento, nossas mentes não são capazes de identificar o melhor caminho.
- **Como tornamos sua solução viável?**
  Em praticamente todos os casos, é possível dividir um GRANDE problema em uma série de PEQUENOS subproblemas.
  - Resolvemos mais rapidamente cada um dos subproblemas e, quando for conveniente, juntamos suas soluções de maneira a encontrar uma solução global: **Divisão e conquista!**
  - Precisamos **modularizar o problema**!

Funções

- No campo da programação...
  - A **modularização de algoritmos** é uma técnica altamente recomendável, que consiste em dividir um algoritmo maior em algoritmos menores (sub-rotinas), tornando o programa principal mais **estruturado**, **organizado** e **refinado**.
- Da modularização, retiramos alguns benefícios:
  - Partes que se repetem são **escritas uma única vez**, facilitando teste e manutenção do código;
  - **Melhor estruturação** do algoritmo, facilitando a depuração de erros e proporcionando melhor documentação;
  - **Bibliotecas de sub-rotinas**, que são disponibilizadas e facilitam a vida do programador.

Funções

- A linguagem C implementa a modularização do código através da criação de **funções**.
  - Enquanto algumas linguagens podem ter significados e representações distintas para “funções”, “sub-rotinas” e “procedimentos”, em C, tudo é tratado simplesmente como **função**.
- **Funções** são blocos de código que implementam uma sequência de **comandos**, recebem um **nome** e podem ser **chamadas** de qualquer parte do programa, quantas vezes forem necessárias, durante sua execução.
  - Uma **função** é um módulo do programa, ou um trecho de código, que possui uma **FUNÇÃO específica**!

Ao Trabalho!

- Vamos ver um exemplo:
  - Escreva um programa que tenha uma **função** que converta graus **Celsius** para **Fahrenheit**. A seguir, teste sua utilização, chamando-a na função main().

Fórmula para Conversão:

$$\frac{C}{5} = \frac{F - 32}{9}$$

```c
#include <stdio.h>

// Função Celsius2Fahr
// Converte um valor em graus Celsius para Fahrenheit
// C/5 = (F-32)/9
float Celsius2Fahr(float celsius)
{
    float fahr;
    fahr = celsius*9/5 + 32;
    return fahr;
}

// Função Principal
int main()
{
    float c, f;

    // Entrada de dados
    printf("Entre com o valor em graus Celsius: ");
    scanf("%f", &c);

    // chamada da função Celsius2Fahr (retorn
    f = Celsius2Fahr(c);

    // Imprime resultado
    printf("%.1f C = %.1f F\n\n", c, f);

    return 0;
}
```

Declaração da função.
Deve ser feita antes da sua utilização, ou chamada.

Definição da função.
Pode ser realizada em qualquer parte do código e define seu comportamento.

Chamada da função.
Retorna um valor do tipo definido pela função.

```
Entre com o valor em graus Celsius: 100
100.0 C = 212.0 F
```

```c
#include <stdio.h>

// Função Celsius2Fahr
// Converte um valor em graus Celsius para Fahrenheit
// C/5 = (F-32)/9
float Celsius2Fahr(float celsius)
{
    float fahr;
    fahr = celsius*9/5 + 32;
    return fahr;
}

// Função Principal
int main()
{
    float c, f;

    // Entrada de dados
    printf("Entre com o valor em graus Celsius: ");
    scanf("%f", &c);

    // chamada da função Celsius2Fahr (retorna float)
    f = Celsius2Fahr(c);

    // Imprime resultado
    printf("%.1f C = %.1f F\n\n", c, f);

    return 0;
}
```

```
Entre com o valor em graus Celsius: 100
100.0 C = 212.0 F
```

Ao Trabalho!

- Como ficaria o fluxo do programa?

```c
float c, f;

// Entrada de dados
printf("Entre com o valor em graus Celsius: ");
scanf("%f", &c);

// chamada da função Celsius2Fahr (retorna float)
f = Celsius2Fahr(c);
```

Chama função, passando parâmetro (tipo *float*).

```c
float Celsius2Fahr(float celsius)
{
    float fahr;
    fahr = celsius*9/5 + 32;
    return fahr;
}
```

```c
// chamada da função Celsius2Fahr (retorna float)
f = Celsius2Fahr(c);

// Imprime resultado
printf("%.1f C = %.1f F\n\n", c, f);

return 0;
```

Devolve controle para o programa principal, retornando o valor calculado (também *float*).

Funções

- Em C, todo o código executável reside dentro de uma **função**.
  - O fluxo principal do programa é sempre escrito dentro da **função main**, portanto, já conhecemos a sintaxe.
- Podemos dizer que os programas em C são combinações de **funções novas**, escritas pelo programador, com **funções pré-existentes**, ou pré-definidas, disponíveis na biblioteca-padrão de C.
  - Já utilizamos muitas vezes funções que estão pré-definidas na **biblioteca-padrão do C**, como printf, scanf, gets, rand, sqrt, strlen, etc.
  - Não é necessário saber como cada uma delas foi escrita. **Basta sabermos utilizá-las**.

Funções

- Duas razões podem ser consideradas as principais para a utilização de funções:
  - **Estruturação dos Programas:** A partir de agora, construiremos os programas a partir de pequenos blocos de código (funções), cada um deles com uma tarefa específica e bem definida. Isso facilita a compreensão do programa como um todo e leva à solução de problemas mais complexos.
  - **Reutilização de Código:** Escrevemos um código uma única vez e podemos utilizá-lo diversas vezes no programa. Por exemplo, se quisermos converter graus Celsius para Fahrenheit em diversas ocasiões no mesmo programa, bastaria chamar a função ao invés de replicar o código (cópia desnecessária).

Funções

- Como declarar uma função?

```
tipo_retorno nome_funcao (lista de parâmetros)
{
        // sequência de declarações e comandos
        return valor_tipo_retorno;
}
```

- O **nome da função** é o identificador através do qual aquele trecho de código será conhecido dentro do programa. Para defini-lo, seguimos as mesmas regras de criação de nomes de variáveis.
- O **tipo de retorno** é o tipo do dado que será retornado à função chamadora.
  - No exemplo anterior, o tipo de retorno era **float**, uma vez que a função retornar um **valor real** que representa, em Fahrenheit, o valor em graus Celsius passado como parâmetro.

Funções

- Como funciona uma função? Veja o exemplo:

```c
#include <stdio.h>

// Calcula quadrado de numero inteiro
int square(int n)
{
    return (n*n);
}

// função pricipal
int main()
{
    int num, sq;
    printf("Entre com o numero: ");
    scanf("%d", &num);

    sq = square(num);

    printf("%d^2 eh %d.\n", num, sq);

    return 0;
}
```

Passagem por Valor

- O código do programa é executado até encontrar a chamada da função.
- O programa é interrompido e o fluxo transferido para a função chamada.
- Os valores da chamada da função são copiados para os parâmetros **(n = num)**.
- Os comandos da função são executados.
- Quando a função termina, o valor de retorno é copiado para a variável que foi escolhida para retorno na função chamadora
- **(sq = n*n)**.

Funções

- Os parâmetros de uma função são informações que gostaríamos de transmitir para dentro da função.
  - As variáveis possuem **escopo**, e somente são válidas dentro das funções que as declararam (veremos mais adiante).
  - Uma variável “num”, declarada na função **main**, não pode ser acessada dentro da função **square**. É necessário que o valor seja “transmitido” através de parâmetros.
  - O caminho de volta é o **valor retornado.**
- Uma função, portanto, receberá todas as informações necessárias para sua tarefa (parâmetros) e devolverá apenas um valor (valor de retorno) à função que a chamar.
  - Chamar funções não é exclusividade da main. Qualquer função pode chamar outras funções, contanto que elas já estejam definidas.

Funções

- Os parâmetros de uma função são listados dentro dos parênteses que seguem seu nome, da seguinte maneira:

```
tipo_retorno nome_funcao (tipo nome1, tipo nome2, tipo nome3, ..., tipo nomeN)
{
        // sequência de declarações e comandos
        return valor_tipo_retorno;
}
```

- Cada um dos parâmetros pode ter seu próprio tipo e deve ter um nome diferente dos demais, que será como o identificaremos no corpo da função. Veja alguns exemplos:

```
int potencia(int base, int expoente);
float media(int num_alunos, float media1, float media2, float trab);
int fatorial (int valor);
void imprime_menu_opcoes();
```

Funções

- Existem ainda alguns casos especiais:
  1. A função não precisa retornar um valor
     Veja o exemplo:

```
// Imprime menu com opções de navegação
void imprime_opcoes()
{
    printf("Sistema de Navegação de Robô \n\n");
    printf("Escolha uma das opções abaixo: \n");
    pritnf("A - Mover para a esquerda \n");
    printf("S - Mover para baixo \n");
    printf("W - Mover para cima \n");
    printf("Digite a opção: ");
}
```

A função somente imprime um “menu”, e não retorna valores. Nesse caso, seu tipo de retorno deve ser definido como void, e não é necessário inserir o comando return ao final da função.

Funções

- Existem ainda alguns casos especiais:
  2. A função não recebe parâmetros
     Veja o exemplo:

```
// Imprime menu com opções de navegação
char imprime_opcoes()
{
    char opcao;
    printf("Sistema de Navegação de Robô \n\n");
    printf("Escolha uma das opções abaixo: \n");
    printf("A - Mover para a esquerda \n");
    printf("S - Mover para baixo \n");
    printf("D - Mover para a direita \n");
    printf("W - Mover para cima \n");
    printf("Digite a opção: ");
    opcao = getchar();

    // retorna opção escolhida
    return opcao;
}
```

Neste caso temos **duas opções**:

1. Deixar a lista de parâmetros vazia, como no exemplo ao lado:
   char **imprime_opcoes**()
2. Colocar void entre parênteses, indicando que a lista de parâmetros é vazia:
   char **imprime_opcoes**(void)

Não significam a mesma coisa!

Funções

- Apesar de as duas declarações estarem corretas, existe uma diferença entre elas:
  - Quando **nenhum parâmetro é especificado** e os parênteses ficam vazios, a função ainda pode ser chamada passando-se parâmetros para ela. O compilador não irá verificar se a função é realmente chamada sem parâmetros, não irá acusar erro, mas a função não terá acesso a eles.
  - Na segunda declaração, com void, nenhum parâmetro é esperado. Nesse caso, o compilador acusará um erro se algum parâmetro for passado. O segundo caso é uma **boa prática de programação**, uma vez que garante a utilização correta da função.

```
// Imprime menu com opções de navegação
char imprime_opcoes(void)
```

Funções

- O corpo da função é a sua parte que define a tarefa que ela irá realizar. É formado por:
  - Uma **sequência de declarações**: variáveis, constantes, vetores, matrizes...
  - E uma **sequência de comandos**: sequenciais, condicionais, laços de repetição, chamadas a outras funções, etc. Segue todas as regras da programação na main(), e retorna um valor ao final.
- Não se esqueça!

  Todos os programas possuem pelo menos uma função: a **main**!

Funções

- A função pode devolver apenas um único valor, e o resultado é devolvido através do comando **return**.

O valor retornado por uma função deve seguir as mesmas regras que são aplicadas a um operador de atribuição.

Por exemplo, um valor **int** não pode ser retornado se o tipo de retorno da função for **char**.

Sempre que o programa encontrar uma instrução **return**, retorna para a instrução que originou a chamada para a função.

A execução de uma chamada à função também termina se não encontrar nenhuma instrução return. Nesse caso a execução continua até a chave final do corpo da função.

Ao Trabalho!

- Tomemos o algoritmo de **cálculo de velocidade média**, que já resolvemos algumas vezes.
  - Como o escreveríamos como função?
  - Quais são os parâmetros de entrada?
  - Qual é o tipo de retorno?

Entrada	Retorno
Distância percorrida (**real**) Tempo total gasto (**real**)	Velocidade média (**real**)

```
// calcula a velocidade media
float velocidade_media(float tempo, float distancia)
```

```c
#include <stdio.h>

// calcula a velocidade media
float velocidade_media(float tempo, float distancia)
{
    float veloc;
    veloc = distancia/tempo;
    return veloc;
}

// Função principal
int main()
{
    float s, t, v;
    printf("Entre com a distancia (m) e tempo (s), separados por espaco: ");
    scanf("%f %f", &s, &t);

    // chamada da funcao
    v = velocidade_media(t, s);

    printf("A velocidade media eh de %.2f ms. \n", v);

    return 0;
}
```

```
Entre com a distancia (m) e tempo (s), separados por espaco: 1000 20
A velocidade media eh de 50.00 ms.
```

Ao Trabalho!

- Agora, escreva uma função que calcule o **fatorial** de um dado número inteiro.
  - Quais são os parâmetros de entrada?
  - Qual é o tipo de retorno?

Entrada	Retorno
Número dado (**inteiro**)	Fatorial do número dado (**inteiro**)

```
// calcula o fatorial
int fatorial(int numero_dado)
```

```c
#include <stdio.h>

// calcula o fatorial
int fatorial(int numero_dado)
{
    int fat = 1, i;

    for(i = 2; i <= numero_dado; i++)
    {
        fat *= i;
    }

    return fat;
}

// Função principal
int main()
{
    int numero, f;
    printf("Entre com o numero: ");
    scanf("%d", &numero);

    // chamada da funcao
    f = fatorial(numero);

    printf("O fatorial de %d eh %d. \n", numero, f);

    return 0;
}
```

```
Entre com o numero: 5
O fatorial de 5 eh 120.
```

Ao Trabalho!

- Elabore uma função que receba três valores inteiros, referentes à horas, minutos e segundos, representando um dado horário do dia. A função deve retornar a **quantidade de segundos** passada desde a meia noite.
  - Quais são os parâmetros de entrada?
  - Qual é o tipo de retorno?

Entrada	Retorno
Horas (**inteiro**) Minutos (**inteiro**) Segundos (**inteiro**)	Total segundos (**inteiro**)

```
// calcula o total de segundos
int total_segundos(int horas, int minutos, int segundos)
```

```c
#include <stdio.h>

// calcula o total de segundos
int total_segundos(int horas, int minutos, int segundos)
{
    int total;
    total = horas*60*60 + minutos*60 + segundos;
    return total;
}

// Função principal
int main()
{
    int h, m, s, total;
    printf("Entre com um horario (hh:mm:ss): ");
    scanf("%d:%d:%d", &h, &m, &s);

    // chamada da funcao
    total = total_segundos(h, m, s);

    printf("Total de segundos desde a meia noite: %d. \n", total);

    return 0;
}
```

```
Entre com um horario (hh:mm:ss): 02:30:15
Total de segundos desde a meia noite: 9015.
```