Pontifícia Universidade Católica
do Rio de Janeiro

Introdução a Programação de Jogos

Aula 05 – Funções

Edirlei Soares de Lima
<elima@inf.puc-rio.br>

visionlab
visualization laboratory

Organização de Código

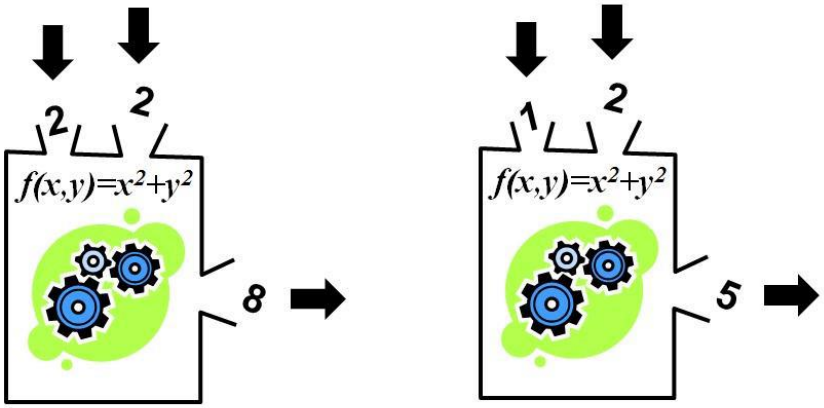
- Um programa representa a implementação de uma solução de um determinado problema.

- **É fundamental o programa seja escrito de forma organizada**, a fim de facilitar a manutenção, o re-uso, a adaptação do código, durante o processo de desenvolvimento ou no futuro.

- Uma maneira de organizar o código é realizando a modularização do programa em **funções**.

Funções

- **Funções** em C são procedimentos que podem ser executados por outras partes do programa (outras funções).



- **São utilizadas para:**
  - Simplificar e organizar o código;
  - Estender a linguagem de programação;

Funções

- **Um programa C é dividido em pequenas funções:**
  - Bons programas são compostos por diversas pequenas funções.
  - Como o próprio nome diz, uma função representa uma funcionalidade.
  - A vantagem de se ter o código modularizado em funções é que o código fica mais fácil de entender, de manter, de atualizar e de reusar.

- Nós já estamos usando funções auxiliares para capturar dados oriundos do teclado (**scanf**) e também para imprimir dados na tela como saída (**printf**).

Criando Novas Funções

Um programa C não pode ter duas funções com o mesmo nome.

```
tipo_de_retorno nome_da_funcao (parametros)
{
   variaveis locais

   instrucoes em C (comandos = expressoes e
   operadores)
}
```

Se uma função não tem retorno colocamos *void*.

Se uma função não tem uma lista de parâmetros colocamos *void* ou apenas o ().

Consiste no bloco de comandos que compõem a função.

visionlab
visualization laboratory

Criando Novas Funções

```
#include <stdio.h>

float celsius_fahrenheit(float tc)
{
  float f;
  f = 1.8 * tc + 32;
  return f;
}

int main (void)
{
  float cels, fahr;
  printf("Digite a temperatura em celsius: ");
  scanf("%f", &cels);
  fahr = celsius_fahrenheit(cels);
  printf("Temperatura em Fahrenheit: %f", fahr);
  return 0;
}
```

/* Podemos usar essa função em qualquer outro programa que precise de uma conversão deste tipo. */

/*As funções devem ser escritas antes de serem chamadas (exceção se usar .h) */

Parâmetros e Valor de Retorno

- Uma função deve ter sua INTERFACE bem definida, tanto do ponto de vista **semântico** quanto do ponto de vista **sintático**:

  - **SEMÂNTICO**: quando projetamos uma função, identificamos sua funcionalidade e com isso definimos que dados de entrada são recebidos e qual o resultado (saída) é produzido pela função.

  - **SINTÁTICO**: os tipos dos dados de entrada e saída são especificados no cabeçalho da função.

Parâmetros e Valor de Retorno

- Exemplo:

```
float celsius_fahrenheit (float tc)
```

Um único parâmetro de entrada

- Exemplo de função que recebe mais de um parâmetro:

```
#include <math.h>

#define PI 3.14159

float volume_cilindro(float r, float h)
{
  float v;
  v = PI * pow(r,2) * h;
  return v;
}
```

Dois parâmetros de entrada que mesmo sendo do mesmo tipo cada um deve ter seu tipo e nome declarados

Uso da função auxiliar
`double pow(double b, double e);`
da biblioteca math.h

Parâmetros e Valor de Retorno

```
int main(void)
{
  float raio, altura, volume;
  printf("Entre com o valor do raio: ");
  scanf("%f", &raio);
  printf("Entre com o valor da altura: ");
  scanf("%f", &altura);

  volume = volume_cilindro(raio, altura);

  printf("Volume do cilindro = ");
  printf("%f", volume);

  return 0;
}
```

Parâmetros e Valor de Retorno

- Uma chamada de uma função pode aparecer dentro de uma expressão maior. Por exemplo, se quiséssemos calcular a metade do volume do cilindro:

```
volume = volume_cilindro(raio,altura)/2.0;
```

- Também pode ser utilizada uma expressão válida na passagem de parâmetros:

```
volume = volume_cilindro(raio, 2*altura);
```

Escopo de Variáveis

- Uma variável declarada dentro de uma função é chamada de **VARIÁVEL LOCAL**:
  - **Ela somente é visível dentro da função que ela está declarada.**
  - Assim que a função termina, os espaços de memória reservados para as suas variáveis locais são liberados e o programa não pode mais acessar esses espaços.

Escopo de Variáveis

- **Variável Local**:
  - Uma função pode ser chamada diversas vezes.
    - Para cada execução da função, os espaços das variáveis locais são automaticamente reservados, sendo então liberados ao final da execução.
  - **Dentro de uma função não se tem acesso a variáveis locais definidas em outras funções.**
  - **Os parâmetros de uma função também são variáveis automáticas com escopo dentro da função.**

```c
#include <math.h>
#define PI 3.14159

float volume_cilindro (float raio, float altura)
{
  float volume;
  volume = PI * pow(raio,2) * altura;
  return volume;
}

int main(void)
{
  float raio, altura, volume;
  printf("Entre com o valor do raio: ");
  scanf("%f", &raio);
  printf("Entre com o valor da altura: ");
  scanf("%f", &altura);

  volume = volume_cilindro(raio, altura);

  printf("Volume do cilindro = ");
  printf("%f", volume);

  return 0;
}
```

Os nomes das variáveis locais são iguais mas a visibilidade é diferente.

Escopo de Variáveis

- Funções em C **recebem VALORES** e **retornam VALORES** (e não nomes de variáveis).

- **Os nomes podem coincidir, mas são variáveis distintas**.

```
float dobra_valor(float x)
{
  x = x * 2;
  return x;
}

int main(void)
{
  float x = 5.0;
  printf("%f ", dobra_valor(x));
  printf("%f ", x);
}
```

Vai escrever 10.0 na tela

Vai escrever 5.0 na tela

Escopo de Variáveis

- **Variável Global:**
  - Declarada fora das funções
  - Vive ao longo de toda execução do programa
  - Visível por todas as funções subsequentes

- **Variável Estática:**
  - Existe durante toda a execução do programa
  - Só é visível dentro da função que a declara

Exercícios

Lista de Exercícios 02 - Funções

http://www.inf.puc-rio.br/~elima/prog-jogos/