visionlab
visualization laboratory

Pontifícia Universidade Católica
do Rio de Janeiro

Introdução a Programação de Jogos

Aula 03 – Funções

Prof. Augusto Baffa
< abaffa@inf.puc-rio.br>

Organização de Código

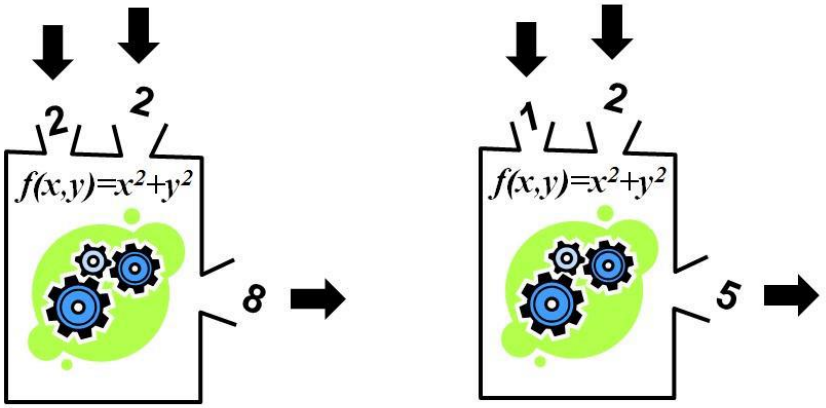
- Um programa representa a implementação de uma solução de um determinado problema.

- **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.

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

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

```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

Exercícios

Lista de Exercícios 01 - Funções

http://www.inf.puc-rio.br/~abaffa/iue1503/

visionlab
visualization laboratory

Pontifícia Universidade Católica
do Rio de Janeiro

Introdução a Programação de Jogos

Aula 03 – Estruturas Condicionais

Prof. Augusto Baffa
< abaffa@inf.puc-rio.br>

Tomada de Decisão

- Até o momento, todas as instruções dos nossos programas eram executadas sequencialmente.
  - Mesmo usando funções elas ainda eram executadas na ordem em que foram codificadas.

- Em geral, precisamos ter maior controle na sequência de instruções que devem ser executadas.

- É fundamental que seja possível tomar diferentes decisões baseado em **condições** que são avaliadas em **tempo de execução.**

Estruturas Condicionais

- **Estruturas condicionais** permitem a criação de programas que não são totalmente sequenciais.

- Com o uso de estruturas condicionais é possível criar **regras** que definem quando uma determinada parte do código deve ser executada.

Estruturas Condicionais

- Em C, a tomada de decisão é construída através do comando ***if***:

```
if (expressão_lógica)
{
   /* Bloco de comandos */
}
```

- Exemplo:

```
if (nota < 5.0)
{
   printf("Reprovado");
}
```

Os comandos do **bloco de comandos** somente são executados se a **expressão lógica** for verdadeira

Estruturas Condicionais

- Também é possível usar o comando **else** para executar algo quando a expressão lógica não é verdadeira:

```
if ( _expressão_booleana_ )
{
   /* Bloco de comandos */
}
else
{
   /* Bloco de comandos */
}
```

Exemplo:

```
if (nota < 5.0)
{
   printf("Reprovado");
}
else
{
   printf("Aprovado");
}
```

Estruturas Condicionais

- Também é possível criar sequencias de comandos **if-else** para a verificação exclusiva de varias condições:

```
if ( _condição_1_ )
{
   /* Bloco de comandos 1 */
}
else if ( _condição_2_ )
{
   /* Bloco de comandos 2 */
}
else if ( _condição_3_ )
{
   /* Bloco de comandos 3 */
}
```

Se a primeira condição resultar em *verdadeiro*, apenas o primeiro bloco de comandos é executado, e as outras condições não são sequer avaliadas.
Senão, se a segunda condição resultar em *verdadeiro*, apenas o segundo bloco de comandos é executado, e assim por diante.

Estruturas Condicionais

- **Exemplo:**

```
if (nota < 3.0)
{
  printf("Reprovado");
}
else if (nota >= 5.0)
{
  printf("Aprovado");
}
else
{
  printf("Em prova final");
}
```

Expressões Booleanas

- Uma expressão booleana é uma expressão que, quando avaliada, resulta no valor falso ou verdadeiro.

- A linguagem C não tem um tipo de dado específico para armazenar valores booleanos.

- Em C, o valor booleano é representado por um valor inteiro:
  - **0** representa **falso**
  - **1** representa **verdadeiro**

Expressões Booleanas

- Uma expressão booleana é construída através da utilização de **operadores relacionais:**

Descrição	Símbolo
Igual a	==
Diferente de	!=
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=

Exemplos:

X = 10 e Y = 5

Expressão	Resultado
(X == Y)	Falso
(X != Y)	Verdadeiro
(X > Y)	Verdadeiro
(X < Y)	Falso
(X >= Y)	Verdadeiro
(X <= Y)	Falso

Todos estes operadores comparam **dois operandos**, resultando no valor 0 (falso) ou 1 (verdadeiro).

Expressões Booleanas

- Expressões booleanas também podem ser combinadas através de **operadores lógicos**.

Operador	Significado	Símbolo em C
Conjunção	E	&&
Disjunção	OU	||
Negação	NÃO	!

Exemplos:

Expressão	Resultado
(X > 0) && (X == Y)	Falso
(X > 0) || (X == Y)	Verdadeiro
!(Y < 10)	Falso

X = 10
Y = 5

Expressões Booleanas

- Operadores lógicos combinam expressões ou valores booleanos, resultando em um valor booleano (0 ou 1).

Conjunção (&&)

Operando 1	Operando 2	Resultado
Falso	Falso	Falso
Falso	Verdadeiro	Falso
Verdadeiro	Falso	Falso
Verdadeiro	Verdadeiro	Verdadeiro

Disjunção (||)

Operando 1	Operando 2	Resultado
Falso	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro

Negação (!)

Operando	Resultado
Falso	Verdadeiro
Verdadeiro	Falso

Expressões Booleanas

- Exemplo 1 (and):

```
...
if (media >= 5.0 && nota1 >= 3.0 && nota2 >=3.0 && nota3 >= 3.0)
{
  printf("Aprovado");
}
...
```

- Exemplo 2 (or):

```
...
if (media < 5.0 || nota1 < 3.0 || nota2 < 3.0 || nota3 < 3.0)
{
  printf("Em prova final");
}
...
```

- Exemplo 3 (not):

```
...
if (!(media < 5.0 || nota1 < 3.0 || nota2 < 3.0 || nota3 < 3.0))
{
  printf("Aprovado");
}
...
```

Estruturas Condicionais - Exemplo

- Crie um programa que converta a nota de um aluno (que varia de 0 a 10) para um conceito (A, B, C, D, ou F). Assuma a seguinte equivalência entre a nota e o conceito:
  - A (9.0 a 10.0)
  - B (8.0 a 8.9)
  - C (7.0 a 7.9)
  - D (5.0 a 6.9)
  - F (menor que 5.0)

```c
#include <stdio.h>

int main (void)
{
  float nota;
  printf("Entre com a nota: ");
  scanf("%f",&nota);
  if (nota >= 9.0)
  {
    printf("A");
  }
  if (nota >= 8.0 && nota < 9.0)
  {
    printf("B");
  }
  if (nota >= 7.0 && nota < 8.0)
  {
    printf("C");
  }
  if (nota >= 5.0 && nota < 7.0)
  {
    printf("D");
  }
  if (nota < 5.0)
  {
    printf("F");
  }
  return 0;
}
```

Estruturas Condicionais

- É importante entender a forma como a linguagem C **avalia expressões booleanas**:
  - Na expressão **(nota >= 8.0 && nota < 9.0)**:
    - O computador primeiro avalia a expressão **nota >= 8.0**. Dependendo do resultado desta expressão, a avaliação da segunda expressão **nota < 9.0**, pode ser omitida.
    - Isto porque se o resultado da primeira expressão for falso, o resultado da expressão lógica como um todo será falso, independente do valor da segunda expressão, pois estamos usando o operador de conjunção (AND).
  - Situação similar ocorre quando usamos o operador de disjunção (OU). Neste caso, se a primeira expressão resultar em verdadeiro, a segunda expressão não é avaliada.

```c
/* solução mais estruturada e mais eficiente */
#include <stdio.h>

int main (void)
{
  float nota;
  printf("Entre com a nota: ");
  scanf("%f",&nota);
  if (nota >= 9.0)
  {
    printf("A");
  }
  else if (nota >= 8.0)
  {
    printf("B");
  }
  else if (nota >= 7.0)
  {
    printf("C");
  }
  else if (nota >= 5.0)
  {
    printf("D");
  }
  else
  {
    printf("F");
  }
  return 0;
}
```

Blocos de Comandos

- Utilizamos chaves ({...}) para delimitar o bloco de comando dos operadores condicionais **if** e **else**:

```
if (nota < 5.0)
{
   printf("Reprovado");
}
```

- Uma variável declarada dentro de um bloco existe enquanto os comandos do bloco estiverem sendo executados. Quando o bloco chega ao fim, as variáveis declaradas dentro dele deixam de existir.

Blocos de Comandos

- Nas construções do comando if, os blocos são importantes para **identificar o conjunto de comandos** cuja execução está submetida à avaliação da expressão booleana.

- No entanto, se um o bloco de comandos for constituído por apenas **um único comando**, as chaves podem ser omitidas.

```
if (nota < 5.0)
   printf("Reprovado");
```

```c
#include <stdio.h>

int main (void)
{
  float nota;
  printf("Entre com a nota: ");
  scanf("%f",&nota);

  if (nota >= 9.0)
    printf("A");
  else if (nota >= 8.0)
    printf("B");
  else if (nota >= 7.0)
    printf("C");
  else if (nota >= 5.0)
    printf("D");
  else
    printf("F");

  return 0;
}
```

Estruturas Condicionais – Exemplo 2

- Crie um programa para calcular as raízes de uma equação do segundo grau.
  - Sabemos que as raízes de uma equação na forma $ax^2 + bx + c = 0$ são dadas por:

$$\frac{-b \pm \sqrt{b^2 - 4*a*c}}{2*a}$$

  - Este seria um problema de codificação direta de uma expressão matemática se não fosse pelo fato das raízes poderem não existir. Na verdade, a raiz quadrada só é definida para valores positivos.

Estruturas Condicionais – Exemplo 2

- Se tentarmos avaliar uma expressão matemática cujo resultado é indefinido, o resultado do programa certamente não será o desejado

- Isto inclui ações como:
  - tentar extrair a raiz quadrada de um número negativo
  - calcular o logaritmo de um número negativo
  - ou mesmo fazer uma divisão por zero

- Por este motivo, devemos avaliar estas expressões apenas após certificarmos que os operandos são válidos

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

int main (void){
  double a, b, c;  /* coeficientes */
  double x1, x2;   /* raízes */
  double delta;
  printf("Entre com os coeficientes (a b c):");
  scanf("%lf %lf %lf", &a, &b, &c);
  if (a == 0.0) {
    printf("Valor de 'a' nao pode ser zero.");
    return 1;
  }
  delta = b*b - 4*a*c;
  if (delta < 0) {
    printf("Raizes reais inexistentes.");
  }
  else if (delta == 0.0) {
    x1 = -b / (2*a);
    printf("Uma raiz real: %f", x1);
  }
  else {
    delta = sqrt(delta);
    x1 = (-b + delta) / (2*a);
    x2 = (-b - delta) / (2*a);
    printf("Duas raizes reais: %f e %f", x1, x2);
  }
  return 0;
}
```

Estruturas Condicionais – Exemplo 3

- Construa um programa que permita calcular o volume de vários tipos de objetos diferentes.
  - O programa deve apresentar um menu para o usuário com os tipos de objetos suportados.
  - O usuário então escolhe a opção desejada, entrar com os dados correspondentes e o programa exibe o volume computado.

- Considere o cálculo de volume dos seguintes objetos:
  - Caixa de lados a, b e c: *volume* = a*b*c
  - Esfera de raio r: *volume* = 4/3*PI*r^3
  - Cilindro de raio r e altura h: *volume* = PI*r^2*h
  - Cone de raio r e altura h: *volume* = 1/3*PI*r^2*h

```c
#include <stdio.h>
#include <math.h>
#define PI 3.1415

void calcula_volume_caixa()
{
  float a, b, c;
  printf("Entre com os lados da caixa:");
  scanf("%f %f %f", &a, &b, &c);
  printf("Volume calculado para caixa: %f", a*b*c);
}

void calcula_volume_esfera()
{
  float r;
  printf("Entre com o raio da esfera:");
  scanf("%f", &r);
  printf("Volume calculado para esfera: %f", 4.0/3.0*PI*pow(r,3);
}

void calcula_volume_cilindro()
{
  float r, h;
  printf("Entre com o raio e altura do cilindro:");
  scanf("%f %f", &r, &h);
  printf("Volume calculado para o cilindro: %f", PI*pow(r,2)*h);
}
```

```
void calcula_volume_cone()
{
  float r, h;
  printf("Entre com o raio e altura do cone:");
  scanf("%f %f", &r, &h);
  printf("Volume calculado para o cone: %f", PI*r*r*h/3.0);
}

int main (void)
{
  int escolha;
  printf("Escolha uma opcao:\n"); /* exibe menu na tela */
  printf("1 - Caixa\n2 - Esfera\n3 - Cilindro\n4 - Cone\n");
  scanf("%d", &escolha);
  if (escolha == 1)
    calcula_volume_caixa();
  else if (escolha == 2)
    calcula_volume_esfera();
  else if (escolha == 3)
    calcula_volume_cilindro();
  else if (escolha == 4)
    calcula_volume_cone();
  else
    printf("Opcao invalida.");

  return 0;
}
```

Exercícios

Lista de Exercícios 02 - Estruturas Condicionais

http://www.inf.puc-rio.br/~abaffa/iue1503/