Lógica de Programação para Engenharia

Modularização e Organização de Código

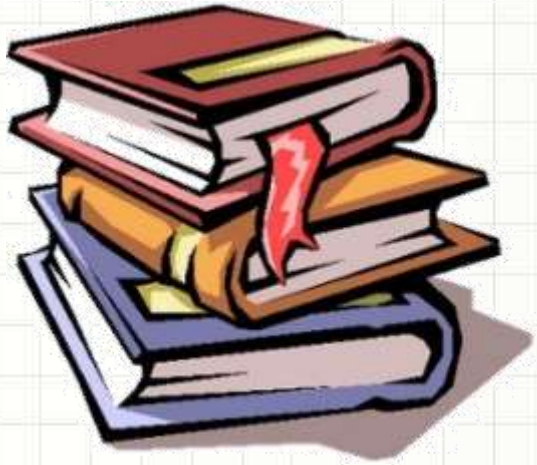
Prof. Dr. Daniel Caetano

2011 - 2

Visão Geral

1. Funções Simples
2. Bibliotecas
3. Funções com Parâmetros
4. A Função Main

Material de Estudo



Material	Acesso ao Material
Notas de Aula	http://www.caetano.eng.br/aulas/lpe/ (Aula 7)
Apresentação	http://www.caetano.eng.br/aulas/lpe/ (Aula 7)
Material Didático	Lógica de Programação – Fundamentos da Programação de Computadores, páginas 7 a 47.

Objetivos

- Entender a utilidade das funções
- Capacitar o aluno para criar suas próprias funções e bibliotecas
- Compreender a função main

- **LISTA 1**

Funções Simples

Funções Simples

- Na aula passada, vimos que PI pode ser calculado como:

$$4 * \text{atan}(1.0)$$

- Será que sempre teremos que usar isso?
- Não seria legal se pudéssemos dar um **nome** para esse código – por exemplo, **pi** – e sempre que precisar usá-lo escrever apenas o nome?

- De fato, é possível!

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
#include <math>
using namespace std;
int main(void)  {
    float A, PI;
    PI = 4 * atan(1.0);    /* Calcula PI */
    A = (60*PI)/180.0;   /* Calcula 60o em rad */
    cout << "Ângulo: " << A;
    getchar();
}
```

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
#include <math>
using namespace std;
int main(void)  {
    float A, PI;
    PI = 4 * atan(1.0);    /* Calcula PI */
    A = (60*PI)/180.0;   /* Calcula 60o em rad */
    cout << "Ângulo: " << A;
    getchar();
}
```

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
#include <math>
using namespace std;

int main(void) {
    float A, PI;
    A = (60*PI)/180.0;  /* Calcula 60o em rad */
    cout << "Ângulo: " << A;
    getchar();
}
```

```
{
    float PI;
    PI = 4.0 * atan(1.0);
}
```

Funciona?

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
#includ      ath>
using names

int mai
    float
    A = (60*PI          /     ula    em rad */
    cout << "Ângul      << A;
    getchar();
}
```

```
{

                    (1.0);
```

NÃO!

Funciona?

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
```

As variáveis de algoritmos diferentes... são diferentes! Precisamos dar um jeito de transferir o resultado de um lugar para o outro

```
    getchar();
}
```

Funciona?

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
#include <math>
using namespace std;

int main(void) {
    float A, PI;
    PI = calcula_pi();
    A = (60*PI)/180.0;
    cout << "Ângulo: " << A;
    getchar();
}
```

```
calcula_pi(void) {
    float PI;
    PI = 4.0 * atan(1.0);
}
```

Ainda dá erro?

Retorno de Dados

- Criamos esse código novo

```
calcula_pi(void)  {
    float PI;
    PI = 4.0 * atan(1.0);
}
```

- Onde está o resultado do cálculo?
- Na variável **PI**, não é?
- Mas o computador não acha isso óbvio!
- Como resolver?

Retorno de Dados

- Vamos resolver com duas palavrinhas:

```
float calcula_pi(void)  {
    float PI;
    PI = 4.0 * atan(1.0);
    return PI;
}
```

- **float** antes do nome: tipo de retorno
- **return**: qual valor deve ser respondido

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
#include <math>
using namespace std;

int main(void) {
    float A, PI;
    PI = calcula_pi();
    A = (60*PI)/180.0;
    cout << "Ângulo: " << A;
    getchar();
}
```

```
float calcula_pi(void) {
    float PI;
    PI = 4.0 * atan(1.0);
    return PI;
}
```

Função

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
#include <math>
using names
float c la_pi(void) {
);
int main
    float
    PI = calcul
    A = (60*P / 180
    cout << "Ângulo " << A;
    getchar();
}
```

Qual a Vantagem?

BIBLIOTECAS

Proj.: modular *Arq.*: extra.hpp

- Em nosso projeto, crie um arquivo chamado **extra.hpp** somente com esse código:

```
#include <math>

float calcula_pi(void) {
    float PI;
    PI = 4.0 * atan(1.0);
    return PI;
}
```

Proj.: modular

Arq.: modular.cpp

```
#include <stdio>
#include <iostream>
using namespace std;
#include “extra.hpp”

int main(void)  {
    float A, PI;
    PI = calcula_pi();
    A = (60*PI)/180.0;
    cout << “Ângulo: ” << A;
    getchar();
}
```

- E modifique o arquivo modular.cpp assim...

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
using namespace std;
#include "extra.hpp"

int main(void) {
    float A, PI;
    PI = calcula_pi();
    A = (60*PI)/180.0;
    cout << "Ângulo: " << A;
    getchar();
}
```

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
using namespace std;
#include "extra.hpp"

int main(void) {
    float A, PI;
    PI = calcula_pi();
    A = (60*PI)/180.0;
    cout << "Ângulo: " << A;
    getchar();
}
```

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
using namespace std;
#include "extra.hpp"

int main(void) {
    float A, PI;
    PI = calcula_pi();
    A = (60*PI)/180.0;
    cout << "Resultado: " << A;
    getchar();
}
```

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
using namespace std;
#include "extra.hpp"
```

Bibliotecas

```
int main(void) {
    float A, PI;
    PI = calcula_pi();
    A = (60*PI)/180.0;
    cout << "Ângulo: " << A;
    getchar();
}
```

Conjunto de funções prontas

Funções com Parâmetros

Funções com Parâmetros

- Funções sem parâmetros são limitadas
- E se quisermos, por exemplo, uma função que arredonde um número?
- Ela precisará receber esse número como parâmetro!
- Opa, essa função já existe: **floor(num)**

```
int arredondado;
arredondado = floor(4.75);
```

- Mas isso sempre arredonda pra baixo! E se quisermos arredondar com a regra normal?

Funções com Parâmetros

- Funções sem parâmetros são limitadas
- F
- **Precisaremos criar uma função e passar o número como parâmetro para ela!**

arredondado = **floor**(4.75);

- Mas isso sempre arredonda pra baixo! E se quisermos arredondar com a regra normal?

Proj.: modular *Arq.*: extra.hpp

- Vejamos... volte ao extra.hpp e adiciona isso:

```
#include <math>

int round(float NUM) {
    int ARRED;
    ARRED = floor(NUM+0.5);
    return ARRED;
}
```

Por que funciona?

Proj.: modular *Arq.*: extra.hpp

- Vejams Voltemos ao extra.hpp

```
#include <math>

int round(float NUM) {
    int ARRED;
    ARRED = floor(NUM+0.5);
}
```

Assinatura

Explica como usar a função: tem o nome round, recebe um parâmetro float e retorna um int

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
using namespace std;
#include "extra.hpp"
int main(void)  {
    float A, B;
    cout << "Digite um número: ";
    cin >> A;
    B = round(A);
    cout << "Arredondado: " << B;
    getchar();
}
```

- Modifique o arquivo modular.cpp assim...

Funções com Parâmetros

- Isso arredonda números para inteiros...
- E se quisermos arredondar com uma casa decimal?

```
float arredonda(float NUM) {
    float ARRED, TEMP, TEMP2;
    TEMP = NUM * 10.0;
    TEMP2 = round(TEMP);
    ARRED = TEMP2 / 10.0;
    return ARRED;
}
```

Funções com Parâmetros

- E se quisermos arredondar com duas casas?

```
float arredonda(float NUM) {
    float ARRED, TEMP, TEMP2;
    TEMP = NUM * 100.0;
    TEMP2 = round(TEMP);
    ARRED = TEMP2 / 100.0;
    return ARRED;
}
```

Funções com Parâmetros

- E se quisermos arredondar com três casas?

```
float arredonda(float NUM) {
    float ARRED, TEMP, TEMP2;
    TEMP = NUM * 1000.0;
    TEMP2 = round(TEMP);
    ARRED = TEMP2 / 1000.0;
    return ARRED;
}
```

Funções com Parâmetros

- E se quisermos arredondar com qualquer número de casas?
- 1 casa → ***10** e **/10**
- 2 casas → ***100** e **/100**
- 3 casas → ***1000** e **/1000**
- n casas → ***?** e **/?**

Funções com Parâmetros

- E se quisermos arredondar com qualquer número de casas?
- 1 casa → ***10** e **/10** → ***10^1** e **/10^1**
- 2 casas → ***100** e **/100** → ***10^2** e **/10^2**
- 3 casas → ***1000** e **/1000** → ***10^3** e **/10^3**
- n casas → ***10^n** e **/10^n**

- Em C: 10^n → **pow(10,n)**

Funções com Parâmetros

- E se quisermos arredondar com qualquer número de casas?

```
float arredonda(float NUM, int casas) {
    float ARRED, TEMP, TEMP2;
    TEMP = NUM * pow(10,casas);
    TEMP2 = round(TEMP);
    ARRED = TEMP2 / pow(10,casas);
    return ARRED;
}
```

- Acrescente esse código no **extra.hpp**!

Proj.: modular *Arq.*: modular.cpp

```
#include <stdio>
#include <iostream>
using namespace std;
#include "extra.hpp"
int main(void) {
    float A, B;
    int CASAS;
    cout << "Digite um número: ";
    cin >> A;
    cout << "Digite um número de casas: ";
    cin >> CASAS;
    B = arredonda(A,CASAS);
    cout << "Arredondado: " << B;
    getchar();
}
```

- E modifique o arquivo **modular.cpp** assim...

A Função Main

Função Main

- Você já deve ter reparado...
- Main é uma função!
- Seu nome é **main** porque é por ela que o computador inicia a execução
- Observe sua definição:

```
int main(void)  {

}
```

Função Main

- Observe sua definição:

```
int main(void) {

}
```

- Retorna **int**: código de erro para o Windows
- **void** indica que ela não usa parâmetros

Conclusões

Resumo

- O uso de funções simplifica a tarefa do programador
- O uso de módulos (.hpp) permite fácil reaproveitamento de código entre programas
- As funções podem receber parâmetros e podem retornar resultados

- **TAREFA!**
  - **Última Semana - Lista 1 (algumas turmas)**

Próxima Aula

- Vamos exercitar alguns programas mais complexos?
  - Vamos programar algumas equações e algoritmos?

PERGUNTAS?

Bom Descanso a Todos!