Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Evandro Chagas Ribeiro da Rosa

Funções

- O objetivo das funções é modularizar o código.
- Uma função é um conjunto de instruções que realiza uma tarefa específica.
- Podemos usar funções para implementar a estratégia de *Dividir para conquistar*.
- Funções também permitem reutilizar código.

Uma função com menos instruções é mais fácil de entender.

```
int main() {
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    instrução;
    return 0;
}
```

```
int main() {
    instrução;
    instrução;
    return 0;
}

void func2() {
    instrução;
    instrução;
}

void func3() {
    instrução;
    instrução;
}

void func4() {
    instrução;
    instrução;
}
```

Definição de Funções

Uma função é composta das seguintes partes:

- Tipo de retorno
- Nome da função
- Lista de parâmetros
- Corpo da função

```
int main(int argc, char** argv) {
  std::cout << "Olá\n";
  return 0;
}
```

Funções void

```
void mostrar_mensagem() {
  std::cout << "Olá da função mostrar_mensagem\n";
}
```

Chamada de Funções

Para chamar uma função, basta colocar `()` após o nome.

```cpp
#include <iostream>

void func_1() {
    std::cout << "Estou na função 1\n";
}

void func_2() {
    std::cout << "Estou na função 2\n";
}

int main() {
    std::cout << "A execução começa no main\n";
    func_1(); // chama a função 1
    func_2(); // chama a função 2
    std::cout << "A execução volta para o main\n";
  return 0;
}
```

Declaração da Função

```cpp
#include <iostream>

// declaração das funções;
void func_1();
void func_2();

int main() {
    std::cout << "A execução começa no main\n";
    func_1(); // chama a função 1
    func_2(); // chama a função 2
    std::cout << "A execução volta para o main\n";
  return 0;
}

void func_1() {
    std::cout << "Estou na função 1\n";
}

void func_2() {
    std::cout << "Estou na função 2\n";
}
```

- Antes de chamar uma função, o compilador precisa saber:
  - O nome da função
  - O tipo do retorno
  - Seus parâmetros
- A declaração da função também é chamada de protótipo da função.

Passando Dados para uma Função

```cpp
#include <iostream>

// declaração da função;
void mostrar_valor(int);

int main() {
    std::cout << "Passando o valor 8\n";
    mostrar_valor(8);
    std::cout << "A execução volta para o main\n";
  return 0;
}

void mostrar_valor(int valor) {
    std::cout << "O valor passado foi "
              << valor << "\n";
}
```

- `int valor` é um *parâmetro* da função `mostrar_valor`.
- Quando chamamos a função `mostrar_valor`, é necessário fornecer um *argumento* do tipo `int`.
- Na declaração da função não é necessário nomear os parâmetros.

Retornando um valor da Função

```cpp
#include <iostream>

// declaração da função;
int soma(int, int);

int main() {
    int resultado = soma(3, 8);
    std::cout << "3 + 8 = ";
    std::cout << resultado << "\n";
    return 0;
}

int soma(int a, int b) {
    int resultado = a + b;
    return resultado;
}
```

- A função `soma` retorna um valor do tipo `int`.
- A instrução `return` é necessária para definir o retorno.
- Podemos usar a chamada de uma função em uma expressão:

```cpp
int resultado = soma(19, 32) / 2;
```

Variáveis Globais

```cpp
#include <iostream>
int valor_global;

void mostrar_valor() {
    std::cout << "O valor_global é "
              << valor_global << "\n";
}
void atualizar_valor(int valor) {
    valor_global = valor;
}

int main() {
    int valor_global  = 10;
    mostrar_valor();
    atualizar_valor(42);
    mostrar_valor();
    std::cout << "O valor_global é "
              << valor_global << "\n";
    return 0;
}
```

Variáveis definidas fora de uma função são chamadas de variáveis globais:

- Qualquer função pode ler ou escrever em uma variável global.
- Variáveis globais são inicializadas em zero.
- Variáveis locais com o mesmo nome *sobrescrevem* variáveis globais.

```cpp
#include <iostream>

const int FECHAR_CONTA = 0;

int fazer_pedido();
float valor_do_item(int);
float calcular_10pc(float);

int main() {
    float subtotal = 0.0;
    while (true) {
        int pedido = fazer_pedido();
        if (pedido == FECHAR_CONTA) {
            break;
        }
        subtotal += valor_do_item(pedido);
    }
    float total = calcular_10pc(subtotal);

    std::cout << "Sua conta ficou no valor de "
              << total << '\n';

    return 0;
}
```

Complete o Código

- Esse código implementa a dinâmica de uma comanda de restaurante.
- Implemente as funções:
  - `fazer_pedido`
  - `valor_do_item`
  - `calcular_10pc`

Argumento por Referência

```cpp
#include <iostream>

void ler_numero_0_100(int &var) {
    do {
        std::cout << "Escreva um número de 0 a 100: ";
        std::cin >> var;
    } while (var < 0 || var > 100);
}

int main() {
    int num1, num2;
    ler_numero_0_100(num1);
    ler_numero_0_100(num2);
    std::cout << "Os números são: "
              << num1 << " "
              << num2 << "\n";
}
```

- Por padrão, o valor do argumento é copiado para a variável de argumento.
- É possível passar argumentos por referência usando uma variável de referência.
- Argumentos passados por referência podem mudar o valor da variável de entrada.

Argumento Padrão

```cpp
#include <iostream>

void add_inplace(int&, int = 1);

int main() {
    int var = 0;
    add_inplace(var);
    add_inplace(var);
    add_inplace(var, 10);
    std::cout << "O valor é: "
              << var << "\n";
}

void add_inplace(int &var, int valor) {
    var += valor;
}
```

- É possível definir um valor padrão para um argumento.
- O valor padrão será usado caso o argumento não seja passado.
- Só é possível definir um valor padrão para os últimos argumentos.
- Não é possível definir um valor padrão para um argumento passado por referência.

Sobrecarga de Funções

```cpp
#include <iostream>

int media(int, int);
double media(double, double);

int main() {
    std::cout << "media(5, 2) = "
              << media(5, 2) << "\n"
              << "media(5.0, 2.0) = "
              << media(5.0, 2.0) << "\n";
    return 0;
}

int media(int a, int b) {
    return (a + b) / 2;
}

double media(double a, double b) {
    return (a + b) / 2;
}
```

Duas ou mais funções podem ter o mesmo nome, desde que tenham parâmetros distintos:

- A *assinatura da função* é o que distingue uma função da outra:
  - `media(int, int)`
  - `media(double, double)`
- O tipo do retorno não faz parte da assinatura da função.

Algoritmos e Programação II

https://evandro-crr.github.io/alg2

Evandro Chagas Ribeiro da Rosa