

# Subrotinas

---

Prof. Thiago Felski Pereira, MSc.

Adaptado: Elisangela Maschio de Miranda

# Definições

---

- Uma subrotina (função ou procedimento) consiste em uma porção de código que resolve um problema muito específico, parte de um problema maior (a aplicação final).
- Pode ser definido como um miniprograma dentro de um programa.
- Blocos de Instruções que realizam tarefas específicas. O código de uma subrotina é carregado uma vez e pode ser executado quantas vezes forem necessárias.

# Definições

---

- Um programa em C++ é uma coleção de funções. Todos os programas se constroem por uma ou mais funções que se integram para criar uma aplicação. Todas as funções contém uma ou mais sentenças C++ e se criam geralmente para realizar uma única tarefa, tal como imprimir a tela, escrever um arquivo ou mudar a cor da tela. Pode-se declarar e executar um número quase ilimitado de funções em um programa C++.

# Vantagens de utilização

---

- Economia de espaço, reduzindo repetições e tornando mais fácil a programação.
- A possibilidade de reutilizar o mesmo código, sem grandes alterações, em outros programas.
- Proporcionam um meio de dividir um projeto grande em pequenos módulos mais manejáveis.
- Ficam mais organizados.

# Tipos de subrotinas

- **Procedimentos:** são subrotinas que não possuem retorno. Em C/C++ são conhecidas como funções sem retorno (e será tratado desta forma nestes slides).

```
1  #include <iostream>
2  using namespace std;
3  #include <locale.h>
4
5  void mensagem () {
6      cout<<"Primeira função sem retorno.";
7  }
8
9  int main () {
10
11      setlocale (LC_ALL, "Portuguese");
12      mensagem();
13
14      return 0;
15  }
16
```

*Função sem retorno*

*Chamada da função*

C:\Users\Usuario\Desktop\Teste\bin\Debug\Teste.exe

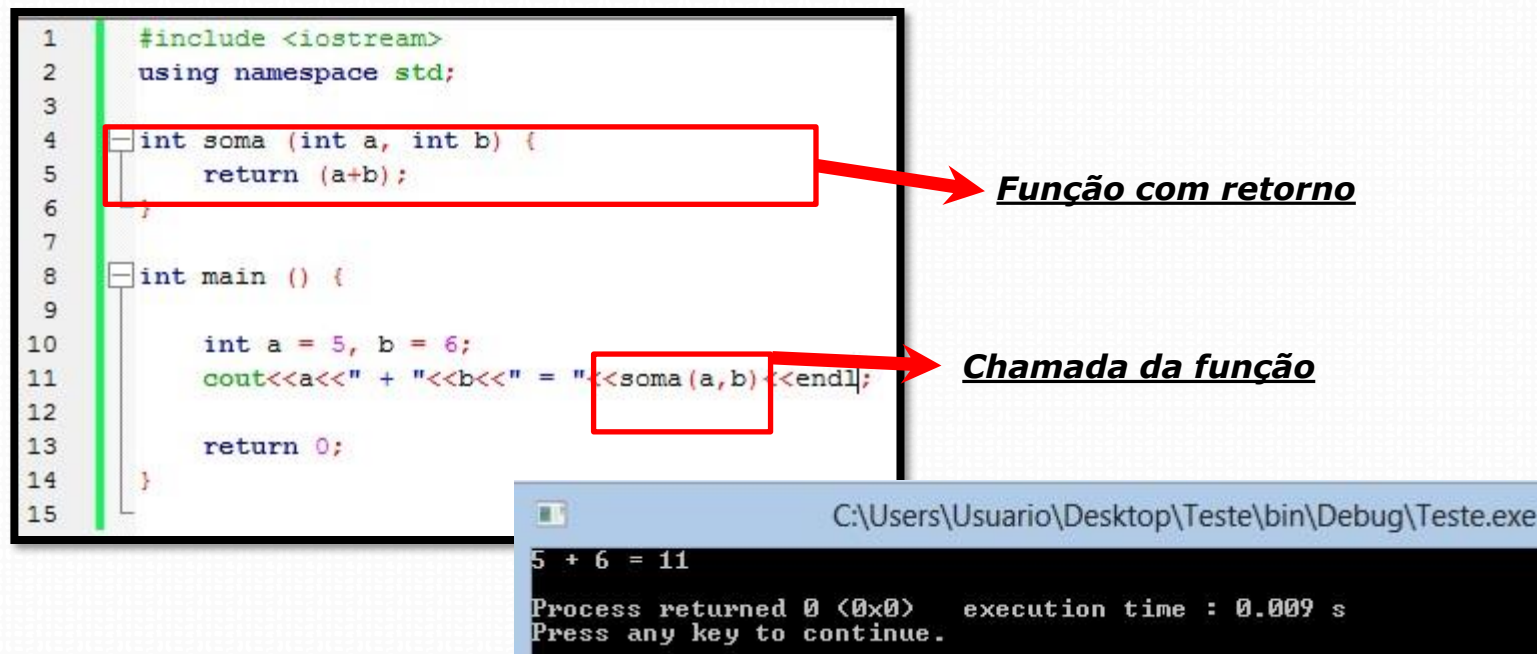
Primeira função sem retorno.

Process returned 0 (0x0) execution time : 0.011 s

Press any key to continue.

# Tipos de subrotinas

- **Funções:** são subrotinas que retornam um valor. Em C/C++ são conhecidas como funções com retorno (e será tratado desta forma nestes slides).



```
1  #include <iostream>
2  using namespace std;
3
4  int soma (int a, int b) {
5      return (a+b);
6  }
7
8  int main () {
9
10     int a = 5, b = 6;
11     cout<<a<<" + "<<b<<" = "<<soma(a,b)<<endl;
12
13     return 0;
14 }
15
```

**Função com retorno**

**Chamada da função**

C:\Users\Usuario\Desktop\Teste\bin\Debug\Teste.exe

5 + 6 = 11

Process returned 0 (0x0) execution time : 0.009 s

Press any key to continue.

# Estrutura de uma função

---

```
tipoDeRetorno nomeDaFunção (listaDeParâmetros) {  
    corpoDaFunção  
    return (expressão);  
}
```

Onde:

- tipoDeRetorno: tipo do valor devolvido pela função, e void caso não haja retorno.
- nomeDaFunção: nome dado a função.
- listaDeParâmetros: variáveis recebidas por parâmetros.
- expressão: valor devolvido pela função.

# Estrutura de uma função

The diagram illustrates the structure of a function in C++ with annotations and a terminal output.

**Code Snippet:**

```
1  #include <iostream>
2  using namespace std;
3  #include <locale.h>
4
5  void leitura (int &x, string letra) {
6      do {
7          cout<<"Valor de "<<letra<<" : ";
8          cin>>x;
9          if (x <= 0) {
10             cout<<"Valor Invalido! Digite novamente."<<endl;
11          }
12      } while (x <= 0);
13  }
14
15  int MDC (int a, int b) {
16      while (a != b) {
17          (a > b) ? a = a - b : b = b - a;
18      }
19      return (a);
20  }
21
22  int main () {
23      setlocale (LC_ALL, "Portuguese");
24      int a, b;
25      leitura(a, "A");
26      leitura(b, "B");
27      cout<<endl<<"O MDC de "<<a<<" e "<<b<<" é "<<MDC(a,b);
28      return 0;
29  }
```

**Annotations:**

- Tipo de retorno:** Points to the return type (`void` for `leitura` and `int` for `MDC`).
- Lista de Parâmetros:** Points to the parameter list (`int &x, string letra` for `leitura` and `int a, int b` for `MDC`).
- Retorno da Função:** Points to the `return (a);` statement in the `MDC` function.
- Chamada de Função:** Points to the function calls `leitura(a, "A");` and `leitura(b, "B");` in the `main` function.

**Terminal Output:**

```
C:\Users\Usuario\Desktop\Teste\bin\Debug\Teste.exe
Valor de A: 12
Valor de B: 20

O MDC de 12 e 20 é 4

Process returned 0 (0x0)   execution time : 2.491 s
Press any key to continue.
```



# Regras para nomes de funções

---

- Segue as regras dos nomes de funções:
  - Nomes significativos.
  - Deve iniciar com uma letra ou `_`(sublinhado), e após pode conter tantas letras, números ou sublinhados quantos deseje o programador.
  - Lembre-se que o C/C++ é sensível a letras maiúsculas/minúsculas, ou seja, se for declarada uma função de nome `Media`, e a chamada estiver `media()`, o compilador não irá reconhecer.

# Tipo de função

---

- Uma função **pode ou não** retornar um valor.
- Em funções em que somente se executa uma tarefa, não havendo necessidade de retornar um resultado para ser utilizado por outra função, diz-se que não há retorno da função.

```
6 void leitura (int vet[TAM]) {  
7     int i;  
8     for (i=0; i<TAM; i++) {  
9         cout<<"Valor na posicao "<<i<<" : ";  
10        cin>>vet[i];  
11    }  
12 }
```

# Tipo da função

- Caso haja necessidade de retornar um valor , deve-se especificar o tipo do retorno da função. O tipo do retorno depende da variável/valor retornado.

O tipo desta função é int pois a variável retornada (elementoMaior) é do tipo int.

```
14 int maior (int vet[TAM]) {  
15     int i, elementoMaior = vet[0];  
16     for (i=1; i<TAM; i++) {  
17         if (elementoMaior < vet[i]) {  
18             elementoMaior = vet[i];  
19         }  
20     }  
21     return (elementoMaior);  
22 }
```

- O tipo da função pode ser de um dos tipos básicos do C/C++, um ponteiro a qualquer tipo do C/C++ ou de um tipo struct.

# Tipo da função

---

- Se o tipo de retorno para uma função é omitido o compilador supõe que o tipo de dado devolvido é int. Mas recomenda-se que mesmo que a função seja int, coloque-se o tipo da mesma, por razões de clareza e consistência.

# Retorno de função

---

- Uma função pode retornar somente **UM** valor.
- O valor retornado pode ser qualquer tipo de dado, exceto uma função, um vetor ou uma matriz.
- **Não se pode retornar Vetores e Matrizes** pois possuem mais que um valor, e uma função retorna somente um valor.
- Uma função pode ter vários return, mas tão logo encontre o primeiro return retorna para a sentença que originou a chamada.

```
return (numero);  
  
return ((a+b)/float(c));  
  
return (false);
```

# Chamada a uma função

- Para que uma função seja executada, é necessário que seja chamada ou invocada.

```
1  #include <iostream>
2  using namespace std;
3  #include <locale.h>
4
5  void funcao1 () {
6      cout<<endl<<"Primeira Função!"<<endl;
7  }
8
9  void funcao2 () {
10     cout<<endl<<"Segunda Função!"<<endl;
11 }
12
13 int main () {
14     setlocale(LC_ALL, "Portuguese");
15     cout<<endl<<"Funcao main(), onde tudo inicia e onde tudo encerra!"<<endl;
16
17     funcao1();
18
19     funcao2();
20
21     return 0;
22 }
```

# Chamada a uma função

---

- Quando se chama uma função, seja no `main()` ou através de outra função, a função chamada irá executar. Ao seu final, retorna aquela função que a chamou.
- Todo programa inicia no `main()`, e encerra no `main()`.
- Quando tem-se uma função sem retorno (`void`), simplesmente a chamada é feita com o nome da função e o conjunto de parâmetros (caso haja).
  - `leitura(a);`
- Em uma função com retorno, deve-se considerar que possui um valor, como se fosse uma variável. Então a chamada deve estar vinculada a outra variável, a um `cout`, por exemplo.

# Chamada a uma função

---

- `cout<<fatorial(num);`
- `if (primo(numero) == true) { ... }`
- `x = MDC(a,b);`
- `C = fatorial(n) / float(fatorial(k) * fatorial(n-k));`



# Chamada a uma função

---

## PRECAUÇÃO

Não se pode definir uma função dentro da outra. Todo código da função deve ser listado sequencialmente durante todo o programa. Antes que apareça o código de uma função, deve surgir a chave de encerramento da função anterior.

# Passagem de parâmetros

---

- Quando se fala em passagem de parâmetros deve-se, em um primeiro momento, entender o escopo de variáveis.
- Existem variáveis globais, locais e escopo de bloco.
- **Variáveis Globais** são aquelas que são declaradas fora de qualquer função e valem para todo o programa.
- **Variáveis Locais** são as declaradas dentro de alguma função. Quando a função é encerrada a variável é destruída e o espaço retorna para a memória.
- **Variáveis de bloco** são as declaradas dentro de um bloco da função e são válidas somente dentro desse bloco. Ao sair dele são destruídas e o espaço retorna para a memória.
- As variáveis locais tem maior prioridade que as globais.

# Passagem de parâmetros

The diagram shows a C++ code snippet with three annotations on the left side, each with an arrow pointing to a specific line of code:

- Variável Global** (red arrow) points to line 5: `int x = 10;`
- Variável Local** (blue arrow) points to line 8: `int resultado;`
- Variável de Bloco (i)** (yellow arrow) points to line 17: `for (int i = 1; i <= 5; i++) {`

```
1  #include <iostream>
2  using namespace std;
3  #include <locale.h>
4
5  int x = 10;
6
7  int calculo (int a, int b) {
8      int resultado;
9      resultado = a * b;
10     return resultado;
11 }
12
13 int main () {
14     setlocale(LC_ALL, "Portuguese");
15
16     int soma = 0;
17     for (int i = 1; i <= 5; i++) {
18         soma = soma + calculo(i, i+2);
19     }
20     cout<<endl<<"Resultado: "<<soma<<" e "<<x;
21
22     return 0;
23 }
```

# Passagem de parâmetros

---

- Parâmetros são informações passadas para as subrotinas realizarem operações sobre elas.
- A Passagem de Parâmetros ocorre na chamada da subrotina, mas para isto, sua definição deve estar preparada especificando o tipo e o nome dos parâmetros. O número de parâmetros e a ordem entre eles deve ser a mesma na chamada e na definição da subrotina.

# Passagem de parâmetros

---

- O nome dos parâmetros corresponde ao nome interno que eles possuem na subrotina e não afetam as variáveis do programa principal, com exceção quando os parâmetros são passados por referência.
- Os parâmetros podem ser passados por valor ou por referência, sendo que:
  - **por valor:** uma cópia do valor é passado para o parâmetro da subrotina.
  - **por referência:** uma referência a posição de memória da variável do programa principal (ponteiro) é passada para o parâmetro.

# Passagem de parâmetros

---

- Quando altera-se o valor de um parâmetro passado por valor, esta alteração não afeta a variável no programa principal.
- Quando altera-se o valor de um parâmetro passado por referência, a variável correspondente no programa principal será alterada.  
Para passagem de parâmetros por referência coloca-se o símbolo & antes do nome da variável.

# Passagem de parâmetros

```
1  #include <iostream>
2  using namespace std;
3  #include <locale.h>
4
5  void parametroValor (int numero) {
6      numero = numero * 2;
7  }
8
9  void parametroReferencia (int &numero) {
10     numero = numero * 2;
11 }
12
13 int main () {
14     setlocale(LC_ALL, "Portuguese");
15
16     int numero = 8;
17
18     cout<<"Numero antes da Passagem por Valor: "<<numero<<endl;
19     parametroValor(numero);
20     cout<<endl<<"Numero depois da Passagem por Valor: "<<numero<<endl;
21     parametroReferencia(numero);
22     cout<<endl<<"Numero depois da Passagem por Referência: "<<numero<<endl;
23
24     return 0;
25 }
```

```
C:\Users\Usuario\Desktop\Teste\bin\Debug\Teste.exe
Numero antes da Passagem por Valor: 8
Numero depois da Passagem por Valor: 8
Numero depois da Passagem por Referência: 16
Process returned 0 (0x0)   execution time : 0.024 s
Press any key to continue.
```

# Lembretes

---

- Todo programa em C/C++ inicia no `main()`, encerra no `main`.
- Se a função tem retorno, a chamada é como se fosse uma variável, vale um valor. Portanto, deve estar ligada a um `cout`, um `if`, um `for`, uma variável, por exemplo.
- Se não possui retorno é simplesmente chamada.
- Passagem de parâmetro e retorno de função são duas situações completamente diferentes. A passagem de parâmetros trata da variável, e o retorno da função faz com que a chamada da função valha um valor.



# Lembretes

---

- A diferença principal entre passagem de parâmetros por valor e por referência (&) é que quando é passada uma variável por valor, a mesma pode ser alterada na função onde é chamada, mas ao final da execução da função não houve modificação do valor. No caso de passagem por referência(&) qualquer alteração realizada na variável na função onde foi chamada, ao finalizar a execução da função, modifica o valor.
- Vetores e Matrizes (arrays) já trabalham com passagem por referência, portanto, não é necessário colocar o &.
- Pode-se retornar somente UM valor, então não é permitido retornar vetores e matrizes.

# Obrigado pela atenção

---

contato: [Felski@univali.br](mailto:Felski@univali.br)

