

BLU3101 – Introdução à Informática para Automação

Modularização

- Funções e Procedimentos -

Prof. Maiquel de Brito

Programação de Computadores

- Linguagem C
 - Subprogramas: procedimentos e funções
 - Variáveis e parâmetros

Subprogramas: Procedimentos e funções

- **Procedimentos e funções** são blocos de código que evitam a repetição excessiva de determinada parte do código de um programa. Também são chamados de **subprogramas**;
- Na linguagem C o bloco de código denominado **main()** é o ponto de partida para a execução do programa;

Subprogramas: Procedimentos e funções

- Anatomia básica de um programa em C
 - Saída: $a + b$ e $c + d$;

Bibliotecas

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

Função principal

```
int main() {
```

Variáveis

```
int a = 3, b = 5, c = 6, d = 10, s;
```

```
s = a + b;
```

Ações repetidas

```
printf("Soma: %d \n", s);
```

```
s = c + d;
```

```
printf("Soma: %d \n", s);
```

```
}
```

- Podemos reescrever este programa criando um subprograma para efetuar a soma de 2 números e mostrar o resultado;

Subprogramas: Procedimentos e funções

- Anatomia básica de um programa em C
 - Saída: a + b e c + d;

Procedimento
para somar e mostrar
dois inteiros

```
#include <stdio.h>

void somar_mostrar(int n1, int n2){
    int s = n1 + n2;
    printf("Soma: %d \n", s);
}
```

Função principal

Variáveis

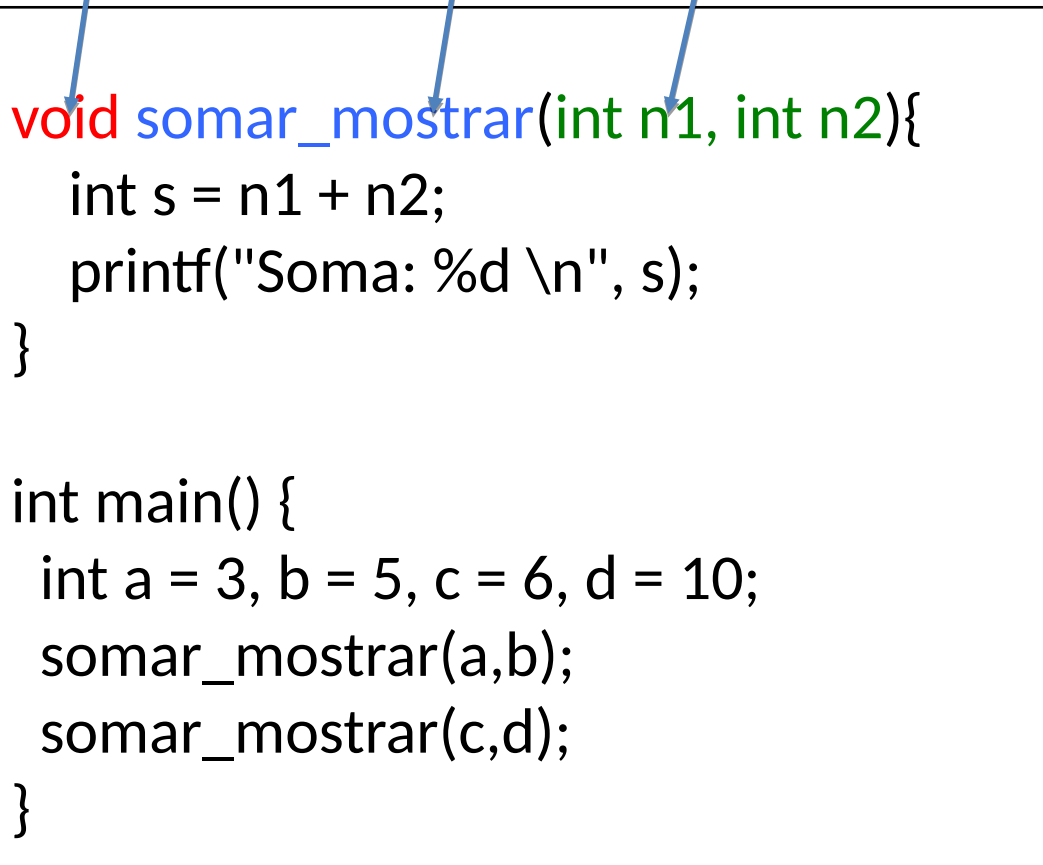
Ações repetidas

A execução será
desviada para o
subprograma

```
int main() {
    int a = 3, b = 5, c = 6, d = 10;
    somar_mostrar(a,b);
    somar_mostrar(c,d);
}
```

Subprogramas: Procedimentos e funções

- Anatomia básica de um subprograma em C
 - Um subprograma é sempre declarado na forma:
tipo_de_retorno **nome** (**parâmetros**) { bloco de código}



```
void somar_mostrar(int n1, int n2){  
    int s = n1 + n2;  
    printf("Soma: %d \n", s);  
}
```

```
int main() {  
    int a = 3, b = 5, c = 6, d = 10;  
    somar_mostrar(a,b);  
    somar_mostrar(c,d);  
}
```

Subprogramas: Procedimentos e funções

- **nome**: identifica o subprograma e deve ser único no contexto do programa.
- Dica: escolher um nome que represente o propósito do subprograma (Ex. **somar_mostrar**)

```
void somar_mostrar(int n1, int n2){  
    int s = n1 + n2;  
    printf("Soma: %d \n", s);  
}
```

Subprogramas: Procedimentos e funções

- **parâmetros**: lista de pares tipo e identificador que descrevem quais serão as entradas necessárias para o subprograma. Dentro do subprograma, os parâmetros são tratados como variáveis locais;
- Dica: evitar o uso do mesmo nome para parâmetros e variáveis do programa principal;

```
void somar_mostrar(int n1, int n2){  
    int s = n1 + n2;  
    printf("Soma: %d \n", s);  
}
```



Subprogramas: Procedimentos e funções

- **tipo_de_retorno**: determina se o subprograma retornará algum valor ao programa principal:
 - void: indica que o subprograma não retorna um valor ao programa principal. Um subprograma que não retorna valor é chamado de **procedimento**;
 - tipos diferentes de void (int, float, char, ...) indicam que o subprograma retornará um valor ao programa principal. Um subprograma que retorna um valor é chamado de **função**. Uma função deverá ter obrigatoriamente a cláusula **return** indicando qual é o retorno da função;

Subprogramas: Procedimentos e funções

- Procedimento x Função

```
void somar_mostrar(int n1, int n2){  
    int s = n1 + n2;  
    printf("Soma: %d \n", soma);  
}  
  
int main() {  
    int a = 3, b = 5, c = 6, d = 10;  
    somar_mostrar(a,b);  
    somar_mostrar(c,d);  
}
```



Solução com
procedimento

Solução com
procedimento e
função

```
int somar(int n1, int n2){  
    int s = n1 + n2;  
    return s;  
}  
  
void mostrar(int n1){  
    printf("Soma: %d \n", n1);  
}  
  
int main() {  
    int a = 3, b = 5, c = 6, d = 10, soma;  
    soma = somar(a,b);  
    mostrar(soma);  
    soma = somar(c,d);  
    mostrar(soma);  
}
```

Subprogramas: Procedimentos e funções

- Variáveis
 - Escopo **global**: existe em todo o programa. A variável x é uma variável global;
 - Escopo **local**: existe apenas dentro do subprograma.
 - As variáveis $n1$, $n2$, s são locais da função `funcao1`.
 - As variáveis a , b , $soma$ são locais da função `main`.

```
#include <stdio.h>

int x = 10;


int funcao1(int n1, int n2){
    int s = (n1 + n2) * x;
    return s;
}

void mostrar(int n1){
    printf("Soma: %d \n", n1);
}

int main() {
    int a = 3, b = 5, soma;
    soma = funcao1(a,b);
    mostrar(soma);
    printf("x: %d \n", x);
}
```


Subprogramas: Procedimentos e funções

- Vetores como parâmetros para funções
 - 3 formas distintas de utilização:
 - Usando ponteiros (conteúdo da BLU3202);
 - Função apenas recebe vetores de um tamanho definido;
 - Função recebe a referência do vetor e o tamanho através de uma segunda variável;



```
void mostrar_vetor(int v[5]){
    int i;
    for (i = 0; i < 5; i++){
        printf("%d ", v[i]);
    }
    printf("\n");
}

int main() {
    int a[5] = {3,6,7,2,1};
    mostrar_vetor(a);
}
```



```
void mostrar_vetor(int v[ ], int n){
    int i;
    for (i = 0; i < n; i++){
        printf("%d ", v[i]);
    }
    printf("\n");
}

int main() {
    int a[5] = {3,6,7,2,1} , b[3] = {1,2,3};
    mostrar_vetor(a, 5);
    mostrar_vetor(b, 3);
}
```

Subprogramas: Procedimentos e funções

- Procedimento: mostrar vetor x Função: somar vetor

```
void mostrar_vetor(int v[5]){
    int i;
    for (i = 0; i < 5; i++){
        printf("%d ", v[i]);
    }
    printf("\n");
}

int main() {
    int a[5] = {3,6,7,2,1};
    mostrar_vetor(a);
    a[3] = 10;
    mostrar_vetor(a);
}
```

```
int somar_vetor(int v[5]){
    int i, s = 0;
    for (i = 0; i < 5; i++){
        s = s + v[i];
    }
    return s;
}

int main() {
    int a[5] = {3,6,7,2,1};
    int soma = somar_vetor(a);
    printf("Soma do vetor: %d \n", soma);
    a[3] = 10;
    soma = somar_vetor(a);
    printf("Soma do vetor: %d \n", soma);
}
```

Subprogramas: Procedimentos e funções

- Passagem de parâmetros para funções
 - 2 formas distintas de utilização:
 - Por **referência**: alterações na variável feitas na função são propagadas para a variável passada como parâmetro (conteúdo da BLU3202);
 - Por **valor**: as variáveis da função são variáveis distintas e ao chamar a função os valores são copiados entre as variáveis, não havendo sincronização dos valores ao final da função;

```
void swap1(int x, int y){  
    int z = x;  
    x = y;  
    y = z;  
}  
int main(){  
    int a = 10, b = 20;  
    swap1(a,b);  
    printf("Valor de a-b: %d - %d\n", a, b);  
}
```

x e y são variáveis locais da função e possuem uma cópia do valor de a e b (parâmetros passados por valor)

Saída:

Valor de a-b: 10 - 20

Valor de a-b: 20 - 10

Criando uma biblioteca

- Funções e procedimentos podem ser armazenados em bibliotecas para serem usados em outros programas
- Ex.: a biblioteca `stdio.h` implementa a função `printf` (entre outras)
- Para criar uma biblioteca:
 - Implementar as funções em um arquivo com extensão `.h` (ex. `funcoes.h`)
 - No programa que for utilizar a biblioteca, declarar `#include "funcoes.h"`. As aspas indicam que o arquivo `funcoes.h` está no mesmo diretório do programa que a utiliza.

Pontos (não menos) importantes

- Vetores e matrizes são sempre passados por referência (sem cópia) para as funções;
- Não é possível em C (sem o uso de ponteiros) retornar um vetor ou matriz como resultado de uma função;
- Uma função pode chamar outra função;
- Algumas linguagens de programação usam nomenclatura diferente:
 - Subprogramas que retornam valor: função
 - Subprogramas que não retornam valor: procedimento

Referências sobre C

- C Como programar. DEITEL, Paul. 6ª Edição;
- Introdução a programação para a engenharia. HOLLOWAY, Paul. 1ª Edição;
- Programar em C (Wikibook)
 - http://pt.wikibooks.org/wiki/Programar_em_C
- Livro aberto: Aprendendo a Programar: Programando na Linguagem C
 - <http://professor.ic.ufal.br/jaime/livros/Aprendendo%20a%20Programar%20Programando%20na%20Linguagem%20C.pdf>