Universidade Positivo

Algoritmos de Programação

Aula 23

Prof.ª Mariane Cassenote

mariane.cassenote@up.edu.br

Conteúdo

- Procedimentos e Funções
- Escopo de variáveis
- Passagem de parâmetros por valor e por referência

Ao final do componente curricular será possível aplicar os conceitos de modularização, procedimentos e funções.

Questões fundamentais

- Quando usar **função** e quando usar **procedimento**?
- Quando usar **variáveis locais** e quanto usar **variáveis globais**?
- Quando usar passagem de parâmetros **por valor** ou **por referência**?

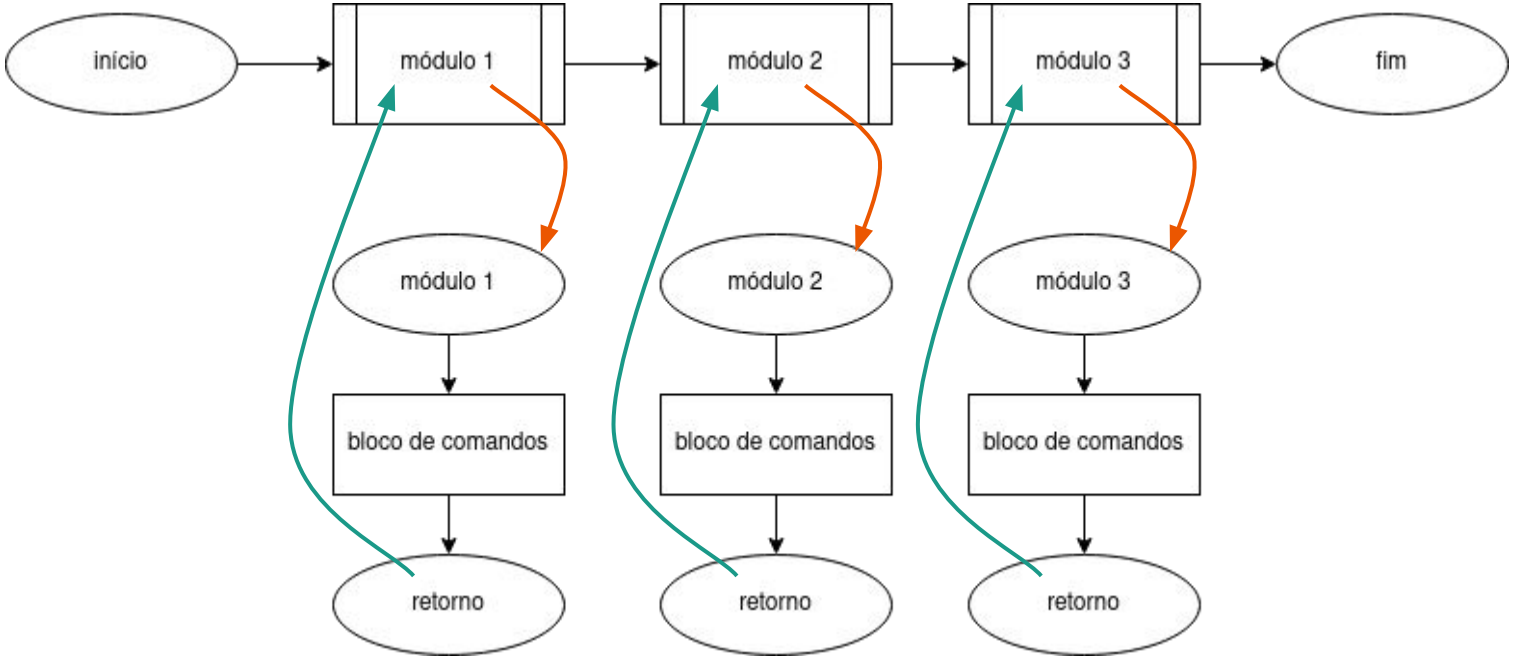
Modularidade

Procedimentos e Funções

Modularidade

- Dividir para conquistar (delegar tarefas)
- Organização e facilidade de manutenção do código
- Reaproveitamento de código (escreve uma vez e chama sempre que necessário)
- Cada módulo pode chamar outros módulos

Modularidade



Modularidade

- Módulos podem ser:
  - **Procedimentos**: não retornam valor
  - **Funções**: retornam valor
- Módulos podem utilizar parâmetros de entrada (argumentos) para troca de informações

Modularidade - em C

Variáveis criadas dentro de um módulo são conhecidas somente por ele (**locais**)

FUNÇÃO SEM RETORNO

```
void <nome> (<parâmetros – opcional>) {
    <declaração de variáveis – opcional>

    <bloco de comandos>
}
```

Se for um **procedimento**, o **retorno** não tem valor.
Se for **função**, tem.

FUNÇÃO COM RETORNO

```
tipo de retorno <nome> (<parâmetros – opcional>) {
    <declaração de variáveis – opcional>

    <bloco de comandos>

    return <valor de retorno>;
}
```

Modularidade - em C

FUNÇÃO SEM RETORNO

```c
#include <stdio.h>
void subtrai (int a, int b) {
    int resultado;
    resultado = a - b;
    printf("Resultado: %d", resultado);
}

int main() {
    int a = 6, b = 2;
    subtrai(a, b);
    return 0;
}
```

FUNÇÃO COM RETORNO

```c
#include <stdio.h>
int subtrai (int a, int b) {
    int resultado;
    resultado = a - b;
    return resultado;
}

int main() {
    int a = 6, b = 2, subtracao;
    subtracao = subtrai(a, b);
    printf("Resultado: %d", subtracao);
    return 0;
}
```

Modularidade

Escreva um programa que exiba o quadrado de um número digitado no teclado.

Escreva uma função para ler esse número e um procedimento para calcular seu quadrado.

Modularidade - em C

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

double leValor() {
  double num; // variável local

  printf("Digite um valor real: ");
  scanf("%lf", &num);

  return num;
}

void quadrado(double valor) {
  double resultado; // variável local
  resultado = pow(valor, 2);

  printf("%.2lf ao quadrado eh %.2lf", valor,
resultado);
}
```

```c
int main() {
  double valor; // variável local
  valor = leValor();
  quadrado(valor);

  return 0;
}
```

O tipo da variável passada por parâmetro só aparece na declaração do módulo, não em sua chamada

```
> ./main
Digite um valor real: 5.2
5.20 ao quadrado eh 27.04>
```

Escopo de variáveis - Variáveis locais

- Também chamadas de variáveis privadas
- São todas as variáveis declaradas em funções / módulos
- As demais funções / módulos não podem usar essas variáveis, uma vez que não conseguem “visualizar” a existência delas

Escopo de variáveis - Variáveis globais

- Também chamadas de variáveis públicas
- Variáveis globais possuem endereços de memória visíveis em todo o programa, até mesmo nas funções / módulos
- São todas as variáveis declaradas logo após o cabeçalho do programa, antes do início de qualquer função, procedimento ou do programa principal

Escopo de variáveis

Desenvolva um programa que leia dois valores inteiros para armazenamento nas variáveis *a* e *b*.

Efetue a troca dos valores, de forma que a variável *a* passe a possuir o valor da variável *b* e a variável *b* passe a possuir o valor da variável *a*.

Apresente os valores das variáveis *a* e *b* após o processamento da troca.

Escopo de variáveis - Exemplo 01 sem modularidade

```c
#include <stdio.h>

int main() {

   int x, a, b; // variáveis locais

   printf("Digite um valor inteiro: ");
   scanf("%d", &a);

   printf("Digite outro valor inteiro: ");
   scanf("%d", &b);

   x = a;
   a = b;
   b = x;

   printf("Trocado: %d %d", a, b);

   return 0;
}
```

```
> ./main
Digite um valor inteiro: 5
Digite outro valor inteiro: 8
Trocado: 8 5>
```

Escopo de variáveis - Exemplo 02 com variáveis globais

```c
#include <stdio.h>

int a, b; // variáveis globais

void troca() {
  int x; // variável local

  x = a;
  a = b;
  b = x;
}
```

→ *(segue no quadro ao lado)*

```c
int main() {

  printf("Digite um valor inteiro: ");
  scanf("%d", &a);

  printf("Digite outro valor inteiro: ");
  scanf("%d", &b);

  troca();

  printf("Trocado: %d %d", a, b);

  return 0;
}
```

Como as variáveis **a** e **b** são globais, não precisamos passá-las por parâmetro para o procedimento **troca()**

```
> ./main
Digite um valor inteiro: 7
Digite outro valor inteiro: 85
Trocado: 85 7>
```

Escopo de variáveis - Exemplo 02 com variáveis globais

```c
#include <stdio.h>

int a, b; // variáveis globais

void troca() {
  int x; // variável local

  x = a;
  a = b;
  b = x;
}
```

→ *(segue no quadro ao lado)*

As variáveis **a** e **b** são válidas até que a execução do programa termine.

A variável **x** é válida somente enquanto o procedimento **troca()** é executado.

```c
int main() {

  printf("Digite um valor inteiro: ");
  scanf("%d", &a);

  printf("Digite outro valor inteiro: ");
  scanf("%d", &b);

  troca();

  printf("Trocado: %d %d", a, b);

  return 0;
}
```

```
> ./main
Digite um valor inteiro: 7
Digite outro valor inteiro: 85
Trocado: 85 7>
```

Escopo de variáveis

- **Cuidado para não criar variáveis globais e locais com o mesmo nome!**
- Variáveis globais são alocadas em memória no início da execução do algoritmo e desalocadas somente no final da execução
- Deve-se priorizar a economia de espaço em memória, então evitar utilizar variáveis globais
- Para priorizar o uso de variáveis locais, é necessário utilizar **passagem de parâmetros**

Passagem de parâmetros

- Comunicação de informações entre módulos
- De modo geral, o ideal é que os módulos recebam **por parâmetro** os valores de que necessitam para realizar seu processamento
- Existem duas maneiras de passar parâmetros para módulos:
    - Por valor
    - Por referência – na próxima aula

Passagem de parâmetros por valor

- Entrega ao módulo uma **cópia** da variável passada como parâmetro
- Qualquer **alteração** nesse parâmetro **não gerará impacto na variável original**
    - Se a `main()` envia o valor de uma variável como parâmetro de uma função e essa função altera o valor dessa variável, a alteração não tem efeito na `main()`

Passagem de parâmetros por valor

```c
#include <stdio.h>

void troca(int a, int b) {

  int x; // variável local

  x = a;
  a = b;
  b = x;

  printf("Trocado: %d %d", a, b);

}
```

→ *(segue no quadro ao lado)*

Para exibir os valores trocados, é necessário realizar a impressão em **`troca()`**

```c
int main() {

  int a, b; // variáveis locais

  printf("Digite um valor inteiro: ");
  scanf("%d", &a);

  printf("Digite outro valor inteiro: ");
  scanf("%d", &b);

  troca(a, b);

  return 0;
}
```

```
> ./main
Digite um valor inteiro: 8
Digite outro valor inteiro: 4
Trocado: 4 8>
```

Passagem de parâmetros por valor

```c
#include <stdio.h>

void troca(int a, int b) {

  int x; // variável local

  x = a;
  a = b;
  b = x;

}
```

→ *(segue no quadro ao lado)*

Nesse caso, as alterações nos valores das variáveis não tem efeito na **`main()`**

```c
int main() {

  int a, b; // variáveis locais

  printf("Digite um valor inteiro: ");
  scanf("%d", &a);

  printf("Digite outro valor inteiro: ");
  scanf("%d", &b);

  troca(a, b);

  printf("Trocado: %d %d", a, b);

  return 0;
}
```

```
> ./main
Digite um valor inteiro: 5
Digite outro valor inteiro: 7
Trocado: 5 7>
```

Questões fundamentais

- Quando usar **função** e quando usar **procedimento**?
- Quando usar **variáveis locais** e quanto usar **variáveis globais**?
- Quando usar passagem de parâmetros **por valor** ou **por referência**?

Para praticar utilizando modularidade

1. Crie um programa que calcule X^Y (X elevado a Y), sendo que os valores de X e Y devem ser números do tipo double, positivos, informados pelo usuário. O resultado deve ser exibido na tela.

2. Elaborar um módulo que apresente a mensagem "Este valor é divisível por 2 ou 3". Deve ser solicitado pelo programa principal o valor a ser verificado. Caso o valor não atenda à condição desejada, o módulo deve apresentar a mensagem "Valor inválido".

Para praticar utilizando modularidade

3. Escreva uma função que recebe uma temperatura em graus Celsius como parâmetro e a converte para graus Fahrenheit. Em seguida, chame a função e imprima o resultado.

4. Escreva uma função que recebe um número inteiro como parâmetro e verifica se ele é um número perfeito. Um número perfeito é aquele cuja soma de seus divisores (excluindo ele mesmo) é igual a ele. Em seguida, chame a função para verificar se um número fornecido pelo usuário é perfeito.

Para praticar utilizando modularidade

5. Escreva uma função que recebe três valores reais como parâmetros: nota1, nota2 e nota3, representando as notas de um aluno em três disciplinas diferentes. A função deve calcular e retornar a média ponderada das notas, considerando os pesos: peso1 = 2, peso2 = 3 e peso3 = 5. Em seguida, chame a função e imprima o resultado.

Universidade Positivo

Algoritmos de Programação

Aula 22

Prof.ª Mariane Cassenote

mariane.cassenote@up.edu.br