Universidade Positivo

Algoritmos de Programação

Aula 24

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 - 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 somar (float a, float b) {
    float resultado;
    resultado = a + b;
    printf("Resultado: %.2f", resultado);
}

int main() {
    float a = 10.2, b = 3.5;
    somar(a, b);
    return 0;
}
```

FUNÇÃO COM RETORNO

```c
#include <stdio.h>
float somar (float a, float b) {
    float resultado;
    resultado = a + b;
    return resultado;
}

int main() {
    float a = 10.2, b = 3.5, soma;
    soma = somar(a, b);
    printf("Resultado: %.2f", soma);
    return 0;
}
```

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 nos 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

- **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

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 referência

- Entrega ao módulo **acesso à variável original**
- Qualquer **alteração** nesse parâmetro está sendo realizada **diretamente na variável original**
- Essas alterações **persistem** mesmo após o término da execução do módulo
- A passagem por referência é uma forma bem estruturada de permitir que um módulo **altere valores de variáveis**

Passagem de parâmetros por referência

- Evita duplicação de variáveis
- Mais eficiente em termos de memória e de tempo de processamento
- Permite a alteração do valor de mais de uma variável
- Pode tornar dispensável o uso de funções com retorno
    - É possível utilizar somente funções sem retorno + passagem de parâmetros por referência

Passagem de parâmetros por referência

```c
#include <stdio.h>

void troca(int *a, int *b) {

  int x;

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

}
```

→ *(segue no quadro ao lado)*

```c
int main() {

  int a, b;

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

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

  printf("Antes da troca: %d %d \n", a, b);

  troca(&a, &b);

  printf("Depois da troca: %d %d \n", a, b);

  return 0;
}
```

```
> ./main
Digite um valor inteiro: 8
Digite outro valor inteiro: 4
Antes da troca: 8 4
Depois da troca: 4 8
>
```

Atenção para o uso de * e &

Passagem de parâmetros por referência

```
programa {

  funcao vazio troca(inteiro &a, inteiro &b) {
    inteiro x
    x = a
    a = b
    b = x
  }
```

→ *(segue no quadro ao lado)*

Atenção para o uso de var na definição do procedimento

```
  funcao inicio() {

    inteiro a, b

    escreva("Digite um valor inteiro: ")
    leia(a)

    escreva("Digite outro valor inteiro: ")
    leia(b)

    escreva("Antes da troca: ", a, " ", b, "\n")

    troca(a, b)

    escreva("Depois da troca: ", a, " ", b, "\n")
  }
}
```

```
Digite um valor inteiro: 2
Digite outro valor inteiro: 3
Antes da troca: 2 3
Depois da troca: 3 2

Programa finalizado. Tempo de execução: 2131 ms
```

Passagem de parâmetros

Escreva um programa que some dois números.

Esse programa deve conter uma função que leia os números do teclado e um procedimento que realize a soma.

Passagem de parâmetros

```c
#include <stdio.h>

int leValor() {

  int valor; // variável local

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

  return valor;
}

void soma(int x, int y, int *resultado) {
  *resultado = x + y;
}
```

→ *(segue no quadro ao lado)*

x e **y** são passados por valor

resultado é passado por referência

```c
int main() {

  int x, y, resultado; // variáveis locais

  x = leValor();
  y = leValor();

  soma(x, y, &resultado);

  printf("Resultado: %d \n", resultado);

  return 0;
}
```

```
> ./main
Digite um valor: 5
Digite um valor: 8
Resultado: 13
>
```

Passagem de parâmetros

x e **y** são passados por valor

resultado é passado por referência

```
programa {

  funcao inteiro leValor() {

    inteiro valor // variável local

    escreva("Digite um valor: ")
    leia(valor)

    retorne valor
  }

  funcao vazio soma(inteiro x, inteiro y, inteiro
  &resultado) {
    resultado = x + y
  }
```

→ *(segue no quadro ao lado)*

```
  funcao inicio() {

    inteiro x, y, resultado // variáveis locais

    x = leValor()
    y = leValor()

    soma(x, y, resultado)

    escreva("Resultado: ", resultado)

  }
}
```

```
Digite um valor: 4
Digite um valor: 3
Resultado: 7
Programa finalizado. Tempo de execução: 1710 ms
```

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. Elaborar um programa que apresente como resultado um número positivo, mesmo que a entrada tenha sido feita com um valor negativo. Use um procedimento com passagem de valor por referência.

2. Crie um programa que leia um número de 1 a 12 e imprima o nome do mês correspondente.

Para praticar utilizando modularidade

3. Um estabelecimento fará uma promoção com descontos nos produtos A e B. Se forem comprados apenas os produtos A ou apenas os produtos B, o desconto será de 10%. Caso sejam comprados os produtos A e B, o desconto será de 15%. O custo de cada produto é, respectivamente, para os produtos A e B, R$10 e R$20. Elaborar um programa que, por meio de módulos, calcule e apresente o valor da despesa do freguês na compra dos produtos. Lembre-se de que o freguês pode levar mais de uma unidade de um determinado produto.

Para praticar utilizando modularidade

4. Escreva um programa que simule a compra de um lanche em uma padaria. O programa deve ser executado enquanto o cliente informar o que deseja comprar e a quantidade desejada. A cada novo item inserido na compra, o programa deve mostrar na tela o subtotal a ser pago. Ao final da execução apresente o valor total da compra.

Universidade Positivo

Algoritmos de Programação

Aula 23

Prof.ª Mariane Cassenote

mariane.cassenote@up.edu.br