Universidade Federal da Bahia - UFBA
Instituto de Matemática - IM
Departamento de Ciência da Computação - DCC
Curso de Bacharelado em Ciência da Computação

MATA40 - Estrutura de Dados Período: 2019.2 Data: 20/08/2019.

Prof. Antonio L. Apolinário Junior

Roteiro do Laboratório 2 - Funções

Objetivos:

- Compreender o conceito de **subprogramação** e **modularidade**;
- Apresentar os mecanismos de definição e uso de funções em Linguagem C;
- Definir escopo de variáveis local e global, no contexto de funções;
- Discutir os conceitos de passagem de parâmetros em Linguagem C, ocm destaque para os mecanismos de passagem por valor e por referência;
- Implementar exemplos de funções em linguagem C.

Conceitos básicos:

Funções:

Em linguagem C, uma função representa um conjunto/bloco de comandos pode ser chamado em qualquer ponto do programa. A definição de uma função segue o padrão:

```
tipo_de_retorno nome_da_função (listagem de parâmetros) {
    bloco de comandos e declarações de variáveis locais
    }
```

Os **parâmetros formais** da função são considerados como variáveis locais a função, que são inicializadas quando do início da execução da função, a partir das variáveis/valores/expressões utilizadas na chamada da função, chamados de **parâmetros reais**.

Parâmetros podem ser passados por valor ou por referência. No primeiro caso, uma cópia dos parâmetros reais é atribuída aos parâmetros formais. Nesse caso, mudanças feitas nos parâmetros formais dentro da função não afetam os parâmetros reais. No caso de passagem de parâmetros por referência, o endereço do parâmetro real é atribuido ao parametro formal, que portanto tem acesso ao mesmo endereço de memória do parametro

real. Portanto, mudanças feitas no parametro formal resultam em mudanças no parametro real.

Um exemplo da diferença de sintaxe dos dois tipos de passagem é mostrado a seguir.

```
void A(int x, int y) { x = 10*y; y = x + 20; }
void B(int* x, int* y) { *x = 10*(*y); *y = *x + 20; }
(……)
int main() {
     int temp1 = 10, temp2 = 20;
(……)
     printf("%d %d\n", temp1, temp2);
     A(temp1, temp2);
     printf("%d %d\n", temp1, temp2);
     B(&temp1, &temp2);
     printf("%d %d\n", temp1, temp2);
(……)
```

Roteiro:

1. Baixe do *Moodle* os códigos fontes desse Laboratório, descompacte e identifique os arquivos gerados.

2. Analise o código fonte do arquivo **`main.c`**. Entenda o que o programa faz e qual a saída esperada. Tire todas as suas dúvidas quanto ao seu funcionamento antes de prosseguir para o próximo item.

3. Analise o mõdulo **`tadVetor.*`**. Esse módulo é responsável por implementar todas as funções utilizadas no programa **`main.c`**. Veja que o programa principal usa a estrutura de dados definido no arquivo **`tadVetor.h`** e as funções implementadas em **`tadVetor.c.`**

4. Para compilar os dois módulos, utilize o script **`Makefile`**. O utilitário **`make`** interpreta esse script e gera os cõdigos objeto e o codigo executavel do programa **`main`**. Para compilar execute no terminal o comando:
   `make`

5. Sua tarefa nesse laboratório é implementar todas as funções especificadas no módulo **`tadVetor.c`** . Os comentários no arquivo **`tadVetor.h`** descrevem o que se espera de cada função. Em caso de dũvida chame o professor/monitor.

Desafio:

1. Crie uma variante desse programa para um **`tadVetorPtos`**, que armazena não de **`floats`** mas de pontos 3D. Cada ponto deve conter as seguintes informações:
   Identificador (ID), um valor inteiro;
   coordenadas (X, Y, Z) em valores reais;
   cor (R, G, B) em valores inteiros no intervalo [0..255].
2. Acrescente operações que calculem:
   - o baricentro do conjunto de pontos;
   - os limites nas 3 direções (X,Y,Z) desse conjunto de pontos;
   - Busque um ponto no conjunto, considerando uma certa tolerancia.