Universidade Federal de São João del-Rei
Departamento de Ciência da Computação
Laboratório de Programação II
Professora Elisa
Professor Guilherme

Instruções para entrega do roteiro:

- Entregue o roteiro apenas no formato *.pdf* com o nome ***Y_roteiroX.pdf***, onde **X é o número do roteiro** e **Y é o número da sua matrícula**. Não serão aceitos outros formatos.
- Inclua nome e matrícula, e mantenha a resolução dos exercícios **ordenada** e **legível**.
- Códigos completos (com int main), compiláveis e executáveis, quando aplicável.

  Para cada um, apresente uma imagem da tela de saída do seu programa.
- Após a data de entrega, a nota da entrega é 0.
- Em caso de dúvidas, procurem os monitores. Haverá um monitor após as aulas de laboratório para tirar dúvidas sobre a lista.

Roteiro 2
Encapsulamento e Análise de Complexidade

Data máxima de entrega: 15/09/2023
(Entrega: pelo SIGAA, na sua turma de laboratório.)

1 Encapsulamento

1.1 Crie uma estrutura chamada **ContaBancaria** para encapsular os detalhes de uma conta bancária. A estrutura deve conter as seguintes informações: *número da conta*, *saldo* e *nome do titular da conta*. Implemente as seguintes funções para interagir com a conta bancária:

- `void criarConta(ContaBancaria* c, int numero, char *titular)`
  - Cria uma nova conta bancária com o número e titular especificados.
  - Inicializa o saldo como zero.
- `void depositar(ContaBancaria *c, double valor)`
  - Deposita o valor especificado na conta.
- `void sacar(ContaBancaria *c, double valor)`
  - Realiza um saque da conta, desde que haja saldo suficiente.
- `double consultarSaldo(ContaBancaria *c)`
  - Retorna o saldo atual da conta.
- `void imprimirInfo(ContaBancaria *c)`
  - Imprime as informações da conta, incluindo número, titular e saldo.

Defina uma *Main* para testar.

1.2 Desenvolva um programa de gerenciamento de catálogo de produtos para uma loja. Crie a estrutura `CatalogoProdutos` para encapsular as informações dos produtos, incluindo nome, preço e quantidade.

Instruções:

- Defina a estrutura `Produto` para encapsular as informações individuais de cada produto, incluindo *nome*, *preço* e *quantidade*.
- A estrutura `CatalogoProdutos` deve conter um array de estruturas `Produto` para armazenar até 100 produtos no catálogo e um inteiro indicando o total de produtos no catálogo.
- Implemente as seguintes funções para manipular o catálogo de produtos (próximo slide).

Funções para o catálogo de produtos:

- `void criarCatalogo(CatalogoProdutos *c);`
  - Cria um catálogo vazio, zerando o total de produtos.
- `void adicionarProduto(CatalogoProdutos *c, char *nome, double preco, int quantidade)`.
  - Adiciona um novo produto ao catálogo.
- `int verificarEstoque(CatalogoProdutos *c, char *nome)`.
  - Verifica a quantidade em estoque de um produto.
- `void imprimirCatalogo(CatalogoProdutos *c)`.
  - Imprime todas as informações dos produtos no catálogo.

Defina uma *Main* para testar.

2 Análise de Complexidade

2.1 Suponha que estamos comparando implementações de ordenação por inserção e ordenação por intercalação na mesma máquina. Para entradas de tamanho n, a ordenação por inserção é executada em $8n^2$ passos, enquanto a ordenação por intercalação é executada em $64n \lg n$ passos. Para quais valores de n a ordenação por inserção é pior do que a ordenação por intercalação?

2.2 Qual é o menor valor de n tal que um algoritmo cujo tempo de execução é $100n^2$ funciona mais rapidamente que um algoritmo cujo tempo de execução é 2^n na mesma máquina?

2.3 O que significa dizer que uma função $g(n)$ é $O(f(n))$?

2.4 O que significa dizer que uma função $g(n)$ é $\Omega(f(n))$?

2.5 Explique por que a declaração *O tempo de execução no algoritmo A é no mínimo O(n^2)* não tem sentido.

2.6 Considere dois algoritmos A e B com funções de complexidade de tempo $a(n) = n^2 - n + 500$ e $b(n) = 47n + 47$, respectivamente. Para quais valores de n o algoritmo A leva menos tempo para executar do que B?

2.7 Considerando a operação ($s = 1$), calcule a complexidade, no pior caso, do trecho de código abaixo:

```
{
int i,j,k,s;
for(i=0; i < N-1; i++)
  for(j=i+1; j < N; j++)
    for(k=1; k < j; k++)
      s = 1;
}
```

2.8 Considerando apenas a operação ($v[i] > MAX$), mostre que o algoritmo *maior* é $O(n)$, $\Omega(n)$ e $\Theta(n)$:

```
int maior(int*v, int n){
    int i, MAX;
    MAX = v[0];
    for(i=1; i<n; i++)
        if(v[i] > MAX)
            MAX = v[i];
    return MAX;
}
```