

UFSJ - Ciências da Computação

Laboratório de Programação 2

Roteiro 2

Nome: Geraldo Arthur Detomi

Exercício 1.1

roteiro_2/conta_bancaria.h

```
1 #ifndef CONTA_BANCARIA_H
2 #define CONTA_BANCARIA_H
3
4 typedef struct ContaBancaria {
5     int numero;
6     double saldo;
7     char titular[50];
8 } ContaBancaria;
9
10 void criarConta(ContaBancaria* c, int numero, char *titular);
11 void depositar(ContaBancaria *c, double valor);
12 void sacar(ContaBancaria *c, double valor);
13 double consultarSaldo(ContaBancaria *c);
14 void imprimirInfo(ContaBancaria *c);
15
16 #endif
```

roteiro_2/conta_bancaria.c

```
1 #include "conta_bancaria.h"
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 void verificar_conta(ContaBancaria *c) {
7     if (c == NULL) {
8         printf("Conta não foi inicializada\n");
9         exit(1);
10    }
11 }
12
13 void criarConta(ContaBancaria* c, int numero, char *titular) {
14     verificar_conta(c);
15
16     c->numero = numero;
17     c->saldo = 0.0;
18     strcpy(c->titular, titular);
19 }
20
21 void depositar(ContaBancaria *c, double valor) {
22     verificar_conta(c);
23
24     c->saldo += valor;
25 }
26
27 void sacar(ContaBancaria *c, double valor) {
28     verificar_conta(c);
29
30     if (c->saldo < valor) {
31         printf("Você não possui saldo suficiente\n");
32         return;
33     }
34
35     c->saldo -= valor;
36     printf("Saque realizado com sucesso!\n");
37 }
38
39 double consultarSaldo(ContaBancaria *c) {
40     verificar_conta(c);
41
42     return c->saldo;
43 }
44
45 void imprimirInfo(ContaBancaria *c) {
46     verificar_conta(c);
47
48     printf("\n\tDados da conta bancaria\n");
49     printf("Numero da Conta: %d\n", c->numero);
```

```

48 |     printf("Titular: %s\n", c->titular);
49 |     printf("Saldo: %.2f\n", c->saldo);
50 | }

```

roteiro_2/1-1.c

```

1 | #include <stdio.h>
2 | #include "conta_bancaria.h"
3 |
4 | int main() {
5 |     ContaBancaria c;
6 |
7 |     criarConta(&c, 1234, "Mario Augusto");
8 |
9 |     imprimirInfo(&c);
10 |
11 |     printf("Realizando deposito de 25,50...\n");
12 |     depositar(&c, 25.50);
13 |
14 |     imprimirInfo(&c);
15 |
16 |
17 |     printf("Tentando realizar saque de 30,50...\n");
18 |     sacar(&c, 30.50);
19 |
20 |     printf("Tentando realizar saque de 15,50...\n");
21 |     sacar(&c, 15.50);
22 |
23 |     printf("Consultado saldo...\n");
24 |     printf("%.2f\n", consultarSaldo(&c));
25 |
26 |     imprimirInfo(&c);
27 |
28 |     return 0;
29 | }

```

Saída do terminal:

```

arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_2 on mainxxx 25-04-
.: ./1-1.out

    Dados da conta bancaria
Numero da Conta: 1234
Titular: Mario Augusto
Saldo: 0.00
Realizando deposito de 25,50...

    Dados da conta bancaria
Numero da Conta: 1234
Titular: Mario Augusto
Saldo: 25.50
Tentando realizar saque de 30,50...
Você não possui saldo suficiente
Tentando realizar saque de 15,50...
Saque realizado com sucesso!
Consultado saldo...
10.00

    Dados da conta bancaria
Numero da Conta: 1234
Titular: Mario Augusto
Saldo: 10.00

```

Exercício 1.2

roteiro_2/catalogo_produtos.c

```

1 | #include "catalogo_produtos.h"
2 | #include <stdio.h>
3 | #include <stdlib.h>
4 | #include <string.h>
5 |
6 | #define MAX 100
7 |
8 | void verificar_catalogo(CatalogoProdutos *c) {
9 |     if (c == NULL) {
10 |         printf("Catalogo de produtos não foi inicializado\n");
11 |         exit(1);
12 |     }
13 | }
14 |
15 | // Cria um catalogo vazio, zerando o total de produtos.
16 | void criarCatalogo(CatalogoProdutos *c) {
17 |     verificar_catalogo(c);

```

```

18
19     c->total = 0;
20 }
21
22 // Adiciona um novo produto ao catálogo.
23 void adicionarProduto(CatalogoProdutos *c, char *nome, double preco,
24                       int quantidade) {
25     verificar_catalogo(c);
26
27     Produto p;
28     strcpy(p.nome, nome);
29     p.preco = preco;
30     p.quantidade = quantidade;
31
32     c->produtos[c->total] = p;
33
34     c->total++;
35 }
36 // Verifica a quantidade em estoque de um produto.
37 // Se não existir o respectivo produto retorna -1
38 int verificarEstoque(CatalogoProdutos *c, char *nome) {
39     verificar_catalogo(c);
40
41     for (int i = 0; i < c->total; i++) {
42         Produto p = c->produtos[i];
43
44         if (strcmp(nome, p.nome) == 0) {
45             return p.quantidade;
46         }
47     }
48
49     return -1;
50 }
51
52 void imprimirProduto(Produto p) {
53     printf("\nProduto nome: %s\n", p.nome);
54     printf("Quantidade: %d\n", p.quantidade);
55     printf("Preço = %.2f \n\n", p.preco);
56 }
57
58 // Imprime todas as informações dos produtos no catálogo.
59 void imprimirCatalogo(CatalogoProdutos *c) {
60     verificar_catalogo(c);
61     for (int i = 0; i < c->total; i++) {
62         Produto p = c->produtos[i];
63
64         imprimirProduto(p);
65     }
66 }

```

roteiro_2/catalogo_produtos.h

```

1 #ifndef CATALOGO_PRODUTOS_H
2 #define CATALOGO_PRODUTOS_H
3
4 typedef struct Produto {
5     char nome[50];
6     double preco;
7     int quantidade;
8 } Produto;
9
10 typedef struct CatalogoProdutos {
11     Produto produtos[100];
12     int total;
13 } CatalogoProdutos;
14
15 // Cria um catálogo vazio, zerando o total de produtos.
16 void criarCatalogo(CatalogoProdutos *c);
17 // Adiciona um novo produto ao catálogo.
18 void adicionarProduto(CatalogoProdutos *c, char *nome, double preco,
19                       int quantidade);
20 // Verifica a quantidade em estoque de um produto.
21 int verificarEstoque(CatalogoProdutos *c, char *nome);
22 // Imprime todas as informações dos produtos no catálogo.
23 void imprimirCatalogo(CatalogoProdutos *c);
24
25 #endif

```

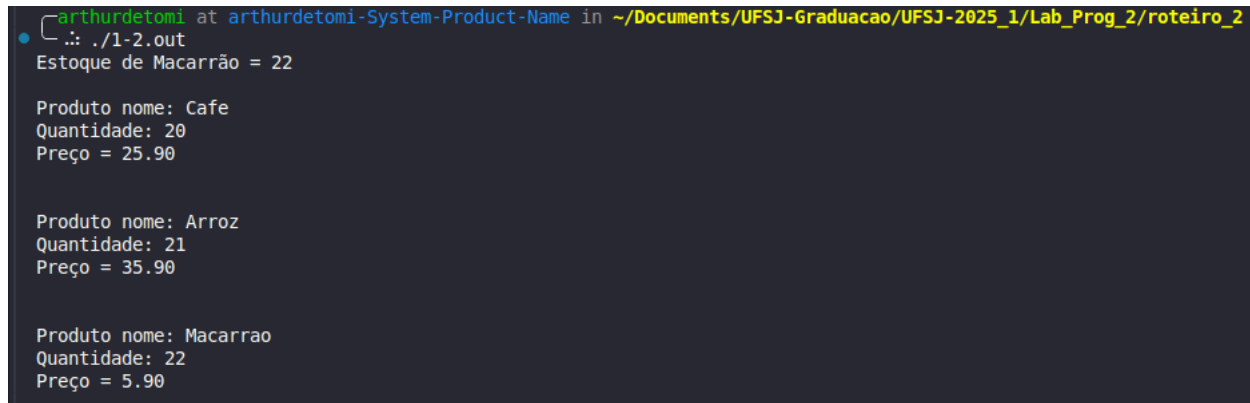
roteiro_2/1-2.c

```

1 #include "catalogo_produtos.h"
2 #include <stdio.h>

```

```
3 |
4 | int main() {
5 |     CatalogoProdutos c;
6 |
7 |     criarCatalogo(&c);
8 |
9 |     adicionarProduto(&c, "Cafe", 25.90, 20);
10 |    adicionarProduto(&c, "Arroz", 35.90, 21);
11 |    adicionarProduto(&c, "Macarrao", 5.90, 22);
12 |
13 |    printf("Estoque de Macarrão = %d\n", verificarEstoque(&c, "Macarrao"));
14 |
15 |    imprimirCatalogo(&c);
16 |
17 |    return 0;
18 | }
```

Saída do terminal:

```
arthurdetomi at arthurdetomi-System-Product-Name in ~/Documents/UFSJ-Graduacao/UFSJ-2025_1/Lab_Prog_2/roteiro_2
● :: ./1-2.out
Estoque de Macarrão = 22

Produto nome: Cafe
Quantidade: 20
Preço = 25.90

Produto nome: Arroz
Quantidade: 21
Preço = 35.90

Produto nome: Macarrao
Quantidade: 22
Preço = 5.90
```

Exercícios de Complexidade 2.1 a 2.8

1 / 1 Geraldo Arthur Detomi
2.1) 2. Análise de complexidade

$$8m^2 > 64m \log_2 m \quad \div 8$$

$$m^2 > 8m \log_2 m \quad \div m$$

$$m > 8 \log_2 m \quad \div 8$$

$$\frac{m}{8} > \log_2 m$$

$$[44, \infty[$$

R = Para m maior ou igual a 44 a ordenação por inserção é pior que a ordenação por intercâmbio.

2.2)

Para $n = 10$

$$100(10)^2 = 10000, 2^{10} = 1024$$

Para $m = 15$

$$100(15)^2 = 22500, 2^{15} = 32768$$

① menor n para o qual $100m^2 < 2^m$ é 15

2.3) Significa que existe uma constante real $c > 0$ e uma constante inteira $n_0 \geq 1$, tal que $0 \leq g(n) \leq cf(n)$ para todo $n \geq n_0$.
Simplificando, significa que a função $g(n)$ é dominada assintoticamente por $\Theta(f(n))$, ou seja $\Theta(g(n))$ é o limite superior de $g(n)$. A notação Θ é utilizada p/ analisar o pior caso de algoritmos.

2.4) A notação Ω descreve o limite inferior assintótico de um algoritmo. Uma notação utilizada para analisar o melhor caso de um algoritmo. Significa que uma função de custo $g(n)$ é $\Omega(f(n))$ se existem duas constantes positivas c e m tais que $n \geq m$, temos $g(n) \geq c \cdot f(n)$

2.5) Não tem sentido, pois a notação O define o limite assintótico superior, ou seja, o pior caso, quando usamos a notação O estamos dizendo que p/ determinado algoritmo seu tempo de execução é no máximo $O(f(n))$, não no mínimo como descrito na declaração.

2.6)

$$a(n) = n^2 - n + 500$$

$$b(n) = 47n + 47$$

$$a(n) < b(n)$$

$$n^2 - n + 500 < 47n + 47$$

$$n^2 - 48n + 453 < 0$$

$$\Delta = b^2 - 4ac \Rightarrow \Delta = (48)^2 - 4(1)(453) \Rightarrow \Delta = 492$$

$$n = \frac{48 \pm \sqrt{492}}{2} \Rightarrow n = \frac{48 \pm 22.18}{2} \quad \begin{matrix} n_1 = 35,09 \\ n_2 = 12,91 \end{matrix}$$

$\frac{48}{2}$

$\frac{22.18}{2}$

$$[12,91 < n < 35,09]$$

$$[13, 35]$$

1 1 d.7)

Total de execuções de $N=1$, sero:

$$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=1}^{j-1} 1$$

- O terceiro loop executa $j-1$ vezes
- O segundo executa em $(N-i)^2$ no pior caso
- O primeiro percorre de $i=0$ até $N-2$ o que adiciona outro fator n

Como possui 3 aninhados (3 loops aninhados) na taxa de crescimento é $O(n^3)$

(2.8)

Para $O(n)$, o loop for executa $n-1$ vezes. Assim o número de comparações cresce linearmente, ou seja, $O(n)$

Para $\Omega(n)$ que seria o melhor caso, o tempo também não pode ser menor que $\Omega(n)$ pois o algoritmo sempre executa $n-1$ vezes

A notação $\Theta(n)$ indica que o tempo é limitado tanto superiormente $O(n)$ como inferiormente $\Omega(n)$ pela mesma expressão, e isso já foi mostrado anteriormente o que prova que é $\Theta(n)$