Nome: _____

Valor:
pontos

Universidade Tecnológica Federal do Paraná – Campus Apucarana
Estruturas de Dados (ED62A) – Lista de Revisão
Prof. Dr. Rafael Gomes Mantovani

Instruções:

- Antes de codificar, esboce em um papel a sequência de passos necessários para criar o seu programa. Isso ajuda a programar a solução;
- Crie um repositório na sua página pessoal do git chamando-o de '*exerciciosRevisao*'. Os exercícios desenvolvidos nessa aula devem ser salvos lá;
- Crie um arquivo .c para cada um dos exercícios e adicione todo o código desenvolvido lá. Por exemplo, na resolução do exercício 01, crie um arquivo chamado 'ex01.c', e uma vez finalizado, faça o upload do mesmo no seu repositório de soluções;

Exercício 1. [**Ponteiros**] Um ponteiro pode ser usado para dizer a uma função onde ela deve depositar o resultado de seus cálculos. Escreva uma função que converta uma quantidade de minutos na notação horas/minutos. A função recebe como parâmetro:

- um número inteiro (**totalMinutos**); e
- os endereços de duas variáveis inteiras, **horas** e **minutos**.

A função deve então atribuir valores às variáveis passadas por referência, de modo que os valores atribuídos respeitem as seguintes condições:

$$\mathbf{minutos} < 60$$
$$60 * \mathbf{horas} + \mathbf{minutos} = \mathbf{totalMinutos}$$

Escreva também a função principal (**main**) que use a função desenvolvida.

Exercício 2. [**Arquivos**] Faça um programa que receba do usuário um arquivo texto. Crie outro arquivo texto de saída contendo o texto do arquivo de entrada original, porém substituindo todas as vogais pelo caractere '*'. Além disso, mostre na tela quantas linhas esse arquivo possui. Dentro do programa faça o controle de erros, isto é, insira comandos que mostre se os arquivos foram abertos com sucesso, e caso contrário, imprima uma mensagem de erro encerrando o programa.

Exercício 3. [**Recursão**] Escreva uma função recursiva para calcular o valor de um número inteiro de base x elevada a um expoente inteiro y.

Exercício 4. [**Alocação Dinâmica**] Faça um programa que leia um valor N e crie dinamicamente um vetor com essa quantidade de elementos. Em seguida, passe esse vetor para uma função que vai ler os elementos desse vetor. Depois, no programa principal, imprima os valores do vetor preenchido. Além disso, antes de finalizar o programa, lembre-se de liberar a área de memória alocada para armazenar os valores do vetor.

Exercício 5. [**Structs, Ponteiros**] Defina um tipo abstrato de dados que irá representar bandas de música. Essa estrutura deve ter o nome da banda, que tipo de música ela toca, o número de integrantes, e em que posição do ranking essa banda está dentre as suas 5 bandas favoritas.

a) Crie uma função para preencher as 5 estruturas de bandas criadas no exemplo passado;

b) Após criar e preencher, exiba todas as informações das bandas/estruturas. Não se esqueça de usar o operador $\rightarrow$ para preencher os membros das structs;

c) Crie uma função que peça ao usuário um número de 1 até 5. Em seguida, seu programa deve exibir informações da banda cuja posição no seu ranking é a que foi solicitada pelo usuário.

Exercício 6. [**Structs, Ponteiros**] Como vimos na aula passada, um baralho normal frequentemente usado em vários jogos para entretenimento pode ser codificado definindo dois tipos abstrato de dados:

- `Carta`: que representa uma carta física do baralho. Possui três atributos: símbolo/valor, o naipe, e uma variável booleana indicando se a carta já foi jogada ou não;
- `Baralho`: uma estrutura que representa um conjunto de `Cartas`.

O código-fonte abaixo mostra possíveis definições para estas estruturas em linguagem C:

```c
typedef struct {
    char valor;
    char naipe;
    bool foiJogada;
} Carta;
```

```c
typedef struct {
    Carta array[54];
} Baralho;
```

Entretanto, apenas a definição dos tipos não garante a modelagem completa do objeto `Baralho`. Para que um baralho seja manipulado adequadamente ele precisa de funções/métodos que mudem seu estado/configuração. Por exemplo, temos que: criar o baralho, adicionando as cartas; consultar a carta do topo ou fundo; embaralhar novamente as cartas; retirar cartas e entregar para os jogadores, etc. A tabela abaixo mostra algumas funções que podem manipular esse tipo de objeto:

Função	**Descrição**
`void` criaBaralho(`baralho` *baralho);	inicia um novo baralho criando todas as cartas nele contido.
`int` cartasNaoJogadas(`Baralho` *baralho);	Consulta o número de cartas disponíveis para jogo.
`Carta` topo(`Baralho` *baralho);	Consulta a carta do topo de um baralho.
`Carta` fundo(`Baralho` *baralho);	Consulta a carta do fundo de um baralho.
`Carta*` carteado(`Baralho` *baralho);	Retorna um array com 3 cartas aleatórias para um jogador;

Tabela 1: Operações básicas com uma baralho

Assim sendo, escreva funções em C para simular os comportamentos listados na Tabela 1. Adicione comandos na função principal que testem e validem todas as funções implementadas.