Lista de exercício

1. *Escreva um programa que calcule a distância entre dois pontos no plano cartesiano. Cada ponto é um par (x,y) de reais (double). Escreva uma estrutura para armazenar cada ponto.*
2. *Considerando a estrutura do exercício anterior, escreva uma função que receba três pontos e determine se os mesmos podem ser usados para formar um triângulo.*
   Dica: Para saber se podemos formar um triângulo com os três pontos precisamos das distâncias entre eles.
3. *Escreva uma função que receba um ponteiro para uma struct do tipo data (dia, mês e ano) e retorne um ponteiro para a data do dia seguinte.*
4. *Considere uma estrutura de conta corrente contendo o nome do titular, o CPF e o valor na conta. Escreva uma função que receba dois ponteiros e realize uma transferência entre as duas contas.*
5. *Fazer um programa para simular uma agenda de telefones. Para cada contato deve-se ter os seguintes dados:*
   - ✦ Nome
   - ✦ E-mail
   - ✦ Telefone (uma estrutura a parte contendo campo para DDD e número)
   - ✦ Data de aniversário (uma estrutura a parte contendo campo para dia, mês, ano)
   - ✦ Observação : Uma linha (*string*) para alguma observação especial.

   1. *Definir a estrutura acima.*
   2. *Declarar a variável agenda (array) de contatos. Usar ponteiro para agenda com tamanho indefinido.*
   3. *Criar o menu para:*
      a) *Definir um bloco de instruções busca por primeiro nome: Imprime os dados da pessoa com esse nome (se tiver mais de uma pessoa, imprime para todas).*
      b) *Definir um bloco de instruções busca por mês de aniversário: Imprime os dados de todas as pessoas que fazem aniversário nesse mês.*
      c) *Definir um bloco de instruções busca por dia e mês de aniversário: Imprime os dados de todas as pessoas que fazem aniversário nesse dia e mês.*
      d) *Definir um bloco de instruções insere pessoa.*
         I. *Caso o array esteja totalmente preenchido, defina uma forma de "aumentar" o tamanho do mesmo.*
            Dica: Crie um novo array maior do que o anterior e copie os elemento para o novo array. Faça o novo array ser o da agenda.
      e) *Definir um bloco de instruções imprime agenda com as opções:*
         i) *imprime nome, telefone e e-mail*
         ii) *imprime todos os dados.*

***DESAFIO**: Permita que a agenda seja armazenada em um arquivo. Defina o formato e modelo de armazenamento dos dados no arquivo.*

Lista ligada

OBS: Os código iniciais estão no final da lista.

6. Implemente as funções declaradas no arquivo `lista_numero.h` que não foram implementadas. São elas:

   ★ `int quantidade();`
   ★ `int primeiro(int * numero);`
   ★ `int ultimo(int * numero);`
   ★ `int removeFim();`

7. Considere a definição de um nó de lista abaixo:

```
struct ListaOrdenada{
 int numero;
 struct ListaOrdenada *proximo;
};
```

   Escreva um programa que realize as seguintes operações numa lista de nós `struct ListaOrdenada`:

   ★ inserir: Insere um número na lista, de forma que o mesmo seja inserido de forma ordenada, onde seu antecessor é menor que ele e seu sucessor é maior que ele.
   ★ remover: remove um número da lista
   ★ buscar: verifica se um número está presente na lista
   ★ tamanho: conta a quantidade de números na lista

8. Considere a seguinte estrutura de um nó:

```
struct Contato {
 int identidade;
 char nome[80];
 char telefone[20];
 struct Contato *proximo;
};
```

   Escreva um programa que implemente uma lista ligada com as seguintes operações:

   ★ inserir: Insere um contato na lista, de forma que o mesmo seja inserido de forma ordenada, onde seu antecessor é menor que ele e seu sucessor é maior que ele. A identidade define a ordem.
   ★ remover: remove um contato da lista partir de sua identidade
   ★ buscar: verifica se um nome está presente na lista. A função deve informa o número de identidade como resultado da busca
   ★ tamanho: conta a quantidade de contatos na lista
   ★ salvar: salva a lista de contatos em um arquivo (pode ser texto ou binário). O nome do arquivo deve ser passado como parâmetro.
   ★ abrir: ler os dados da agenda de um arquivo. O nome do arquivo deve ser passado como parâmetro

9. Desenvolva um programa capaz de manipular mais de uma lista de números/contatos. O ponteiro para o início da lista deve ser passado como parâmetro a partir do programa principal. **IMPORTANTE: As operações de inserção e remoção devem retornar um ponteiro para o início da lista, já que uma remoção ou inserção pode mudar o início.**

Arquivos iniciais de uma lista de números.

IMPORTANTE: O ponteiro para o inicio da lista está definido no arquivo lista_numeros.c. Esta decisão limita a quantidade de lista do programa em apenas 1. Essa decisão foi tomada apenas para fins didáticos, onde o objeto destes exercícios é entender as operações sobre uma lista (inserção, remoção e busca).

lista_numeros.h

```c
#ifndef LISTA_NUMEROS_H_
#define LISTA_NUMEROS_H_

struct ListaNumero {
  int numero;
  struct ListaNumero *proximo;
};

/* Insere numero no início da lista
 * retorna 1 se a inserção ocorrer sem problemas, -1 caso contrário
 */
int inserirInicio(int numero);

/* Insere numero no fim da lista
 */
int inserirFim(int numero);

/* Remove o número do início da lista
 * retorna 1 se a remoção ocorrer sem problemas, -1 caso contrário
 */
int removeInicio();

/* Remove o número do fim da lista
 * retorna 1 se a remoção ocorrer sem problemas, -1 caso contrário
 */
int removeFim();

/* Busca numero na lista
 * retorna 1 se ele estiver na lista, -1 caso contrário.
 */
int buscaNumero(int numero);

/* Conta quantidade de elementos na lista
 */

int quantidade();

/* Coloca no parâmetro 'int * numero' o primeiro número da lista
 * retorna 1 se o número existir, -1 caso a lista esteja vazia.
 */

int primeiro(int * numero);

/* Coloca no parâmetro 'int * numero' o último número da lista
 * retorna 1 se o número existir, -1 caso a lista esteja vazia.
 */

int ultimo(int * numero);

/* Mostra todos os números da lista, um por linha
 */

void mostraNumeros();

#endif /* LISTA_NUMEROS_H_ */
```

lista_numeros.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "lista_numeros.h"

struct ListaNumero *inicio = NULL;

/* Insere numero no início da lista
 * retorna 1 se a inserção ocorrer sem problemas, -1 caso contrário
 */
int inserirInicio(int numero) {
  struct ListaNumero *novo;
  int retorno = 1;
  novo = (struct ListaNumero*) malloc(sizeof(struct ListaNumero));
  if (novo == NULL) { /* Verifica se a memória foi alocada */
          retorno = -1;
  } else {
          novo->numero = numero;
          novo->proximo = inicio;
          inicio = novo;
  }
  return retorno;
}

/* Insere numero no fim da lista
 */
int inserirFim(int numero) {
  struct ListaNumero *novo, *ultimo;
  int retorno = 1;
  novo = (struct ListaNumero*) malloc(sizeof(struct ListaNumero));
  if (novo == NULL) { /* Verifica se a memória foi alocada */
          retorno = -1;
  } else {
          novo->numero = numero;
          novo->proximo = NULL; /* O novo é o fim da lista */
          if (inicio == NULL) { /* Se a lista estiver vazia */
                  inicio = novo; /* o novo será o único elemento da lista */
          } else { /* Se ja houver elementos na lista */
                  /* Percorre a lista para encontrar o último elemento */
                  ultimo = inicio;
                  while (ultimo->proximo != NULL)
                          ultimo = ultimo->proximo;
                  ultimo->proximo = novo; /* Atualiza referência do último */
          }
  }
  return retorno;
}
```

```c
/* Remove o número do início da lista
 * retorna 1 se a remoção ocorrer sem problemas, -1 caso contrário
 */
int removeInicio() {
  struct ListaNumero *velho;
  int retorno = 1;
  velho = inicio;
  if (velho == NULL) {
        retorno = -1;
  } else {
        inicio = inicio ->proximo;
        free(velho);
  }
  return retorno;
}

/*Remove o número do fim da lista
 * retorna 1 se a remoção ocorrer sem problemas, -1 caso contrário
 */
int removeFim() {

  return -1;
}

/* Busca numero na lista
 * retorna 1 se ele estiver na lista, -1 caso contrário.
 */
int buscaNumero(int numero) {
  struct ListaNumero * atual;
  int retorno = -1;
  atual = inicio;
  while (atual != NULL) {
        if (atual->numero == numero) {
              retorno = 1;
              break;
        }
        atual = atual->proximo;
  }
  return retorno;
}

/* Conta quantidade de elementos na lista
 */
int quantidade() {
  return 0;
}

/* Coloca no parâmetro 'int * numero' o primeiro número da lista
 * retorna 1 se o número existir, -1 caso a lista esteja vazia.
 */

int primeiro(int * numero) {
  return -1;
}
```

```
/* Coloca no parâmetro 'int * numero' o último número da lista
 * retorna 1 se o número existir, -1 caso a lista esteja vazia.
 */

int ultimo(int * numero) {
  return -1;
}

/*
 * Mostra todos os números da lista, um por linha
 */

void mostraNumeros() {
  struct ListaNumero * atual;
  atual = inicio;
  printf("Lista de todos os números cadastrados\n\n");
  while (atual != NULL) {
        printf("   %d\n", atual->numero);
        atual = atual->proximo;
  }
}
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "lista_numeros.h";

void printMenu() {
    printf("\n");
    printf("*** Programa exemplo de lista ligada de números inteiros ***\n\n");
    printf("    1 - Inserir número no início da lista\n");
    printf("    2 - Inserir número do fim da lista\n");
    printf("    3 - Remover número no início da lista\n");
    printf("    4 - Remover número do fim da lista\n");
    printf("    5 - Buscar um número na lista\n");
    printf("    6 - Mostra a quantidade de elementos existentes na lista\n");
    printf("    7 - Mostra o primeiro número da lista\n");
    printf("    8 - Mostra o último número da lista\n");
    printf("    9 - Lista todos os números\n");
    printf("\n");
    printf(">>> Digite uma opção, ou 0 (zero) para sair: ");

}

int main(int argc, char **argv) {
    int fim, opcao, numero;
    fim = 0;
    while (!fim) {
        printMenu();
        scanf("%d", &opcao);
        switch (opcao) {
        case 0:
            fim = 1;
            break;
        case 1:
            printf("Digite uo número a ser inserido: ");
            scanf("%d", &numero);
            if (inserirInicio(numero) == 1) {
                printf("Numero %d inserido no início da lista\n", numero);
            } else {
                printf("Erro ao inserir no início da lista\n");
            }
            break;
        case 2:
            printf("Digite uo número a ser inserido: ");
            scanf("%d", &numero);
            if (inserirFim(numero) == 1) {
                printf("Numero %d inserido no fim da lista\n", numero);
            } else {
```

```
                        printf("Erro ao inserir no fim da lista\n");
                }
                break;
        case 3:
                if (removeInicio() == 1) {
                        printf("Numero removido no início da lista\n");
                } else {
                        printf("Erro ao remover do início da lista\n");
                }
                break;
        case 4:
                printf("Opção não implementada!!!!\n\n");
                break;

        case 5:
                printf("Digite uo número a ser buscado na lista: ");
                scanf("%d", &numero);
                if (buscaNumero(numero) == 1) {
                        printf("Numero %d está presente na lista\n", numero);
                } else {
                        printf("A lista não contém o número %d\n", numero);
                }
                break;
        case 6:
                printf("Opção não implementada!!!!\n\n");
                break;
        case 7:
                printf("Opção não implementada!!!!\n\n");
                break;
        case 8:
                printf("Opção não implementada!!!!\n\n");
                break;
        case 9:
                mostraNumeros();
                break;
        default:
                printf("ATENÇÃO: digite uma opção válida!\n");
                break;
        }

    }

    return 0;
}
```