UNIVERSIDADE FEDERAL DO AGRESTE DE PERNAMBUCO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

IDENTIFICAÇÃO	
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS I	**CÓDIGO:**
PROFESSOR: RENÊ GUSMÃO	**PERÍODO: 2023.1**

Trabalho 1 – Listas sequenciais

Nesta atividade, você deve implementar um sistema simplificado para um dos problemas definidos abaixo. O objetivo é utilizar listas sequenciais para modelar e implementar o Tipo Abstrato de Dados (TAD) definido para o problema, isto é, as estruturas que serão utilizadas e as operações que manipulam essas estruturas.

1. Gestão de alunos;
2. Gestão de tarefas;
3. Gestão de estoque;
4. Gestão de biblioteca.

O sistema deve armazenar os dados cadastrados utilizando **alocação dinâmica de memória e arquivos.** O sistema deve ser capaz de armazenar, inicialmente, 10 registros (alunos, tarefas, produtos ou livros) e aumentar a capacidade para 10 novos registros sempre que necessário com alocação dinâmica.

1. O sistema deve ter as funcionalidades descritas abaixo. **Cada funcionalidade deve ser implementada usando uma função.** O código deve estar comentado, ter tabulação correta e seguir boas praticas de programação. Além disso, deve-se definir um arquivo .h para definição dos protótipos das funções.

1.1. **Menu de opções** - O menu imprime mensagens com opções das funcionalidades descritas abaixo (itens 2 até 8) e o usuário selecionará o índice da opção desejada; - O menu será apresentado sempre que o usuário finalizar uma funcionalidade; - O sistema terá final quando o usuário selecionar a opção de “Sair”; - Opções inválidas devem gerar um alerta de erro;

1.2. **Inserção** - Funcionalidade para inserção de apenas um registro. Cada registro deve possuir identificador único. O sistema deve aumentar a capacidade de armazenamento em tempo de execução usando alocação dinâmica de memoria; - Alertas de erro devem ser gerados (exemplos: registros com matrículas iguais, valores inválidos, etc). Esta funcionalidade também deve ser implementada de forma que os registros podem ser incluídos no inicio da lista, no fim ou em posição informada, isto é, três funções de inserção devem ser implementadas. A inserção em uma determinada posição fará com que os registros da posição escolhida até o fim da lista sejam deslocados uma posição a direita;

1.3. **Remoção –** O identificador deve ser informado (remoção por valor); - Essa funcionalidade disponibilizará espaço para um novo registro; - Alertas de erro devem ser gerados (exemplos: identificador não encontrado, sistema sem registros cadastrados). A remoção de um registro em uma determinada posição (remoção por posição) fará com que todos os registros a direita da posição escolhida sejam deslocados uma posição a esquerda;

1.4. **Listagem** - Exibir os registros ativos em um determinado momento. A depender do sistema, outras funções para listagem devem ser consideradas. Para a gestão de alunos, também é necessário ter funções para listar alunos aprovados, reprovados, etc. Outro exemplo pode ser dado para a gestão de tarefas, onde a listagem também pode ser feita para tarefas concluídas, tarefas em andamento, tarefas concluídas, etc. No caso da gestão da biblioteca, deve ser possível listar os livros disponíveis, emprestados, livros por editora e por ano. Outra possibilidade é a de exibir os dados de um único registro a partir de seu identificador, caso este esteja cadastrado;

2. **Arquivo texto:** um arquivo de texto deverá ser criado e manipulado através das funções. O arquivo terá que ser acessado em todas as funcionalidades que necessitem inserir, alterar, remover ou apresentar os dados. Todas as informações inseridas, alteradas ou removidas deverão estar refletidas no arquivo;

2.1. **Requisitos mínimos:** este projeto deverá usar obrigatoriamente constantes, variáveis globais, variáveis locais, estruturas (struct), ponteiros, arquivo de texto e módulos. **O TAD e as operações definidas para manipular a struct (inserção, remoção, inicialização, listagem dos elementos, tamanho) devem ser definidas no arquivo .h e implementadas no .c correspondente**. A função *main* deverá ser implementada em outro arquivo .c que irá apenas utilizar o TAD através da inclusão da biblioteca .h criada.

Prazo: Todos os alunos deverão enviar o código até o dia **29/11/2023** no ambiente virtual da turma no *Classroom*. A atividade poderá ser feita em dupla ou individualmente.

Política para Plágio: Alunos que forem flagrados compartilhando código com colegas ou utilizando código de outros autores receberão nota 0 (**ZERO**) na atividade.

Observação: Abaixo são descritas algumas estruturas para modelar os registros de cada problema. A critério dos alunos, as estruturas podem sofrer pequenas modificações.

1. Gestão de alunos

Os dados abaixo devem ser informados e armazenados sobre cada aluno. Deve-se criar um **struct** para representar o aluno contendo as variáveis descritas abaixo.

Tipo	Variável	Descrição
Inteiro	matrícula	Formato: 20230XX Ex: 2023001, 2023002,etc.
Real	notas	Médias do aluno em 7 disciplinas
String	nome	Nome do aluno
Inteiro	código	Código da turma
Inteiro	faltas	Quantidade de faltas nas 7 disciplinas

Exemplos de estruturas para o aluno e lista de alunos:

```
typedef struct{
    int matricula;
    float notas[7];
    char nome[50];
    int codTurma;
    int faltas[7];
}aluno;
```

```
typedef struct{
    aluno *alunos;
    int qtdAlunos;
    int id;
}turma;
```

2. Gestão de biblioteca

Os dados abaixo devem ser informados e armazenados sobre cada livro. Deve-se criar um **struct** para representar o livro contendo as variáveis descritas abaixo.

Tipo	Variável	Descrição
Inteiro	id	Identificador do livro
String	autores	Autores do livro

String	título	Título do livro
String	editora	Editora do livro
Inteiro	ano	Ano de lançamento
Inteiro	situação	Livro disponível

Exemplos de estruturas para o livro e lista de livros:

```
typedef struct{
    int id;
    char autores[100];
    char titulo[50];
    char editora;
    int ano;
    int situacao;
}livro;
```

```
typedef struct{
    livro *livros;
    int qtdLivros;
}biblioteca;
```

3. Gestão de tarefas

Os dados abaixo devem ser informados e armazenados sobre cada livro. Deve-se criar um **struct** para representar cada tarefa contendo as variáveis descritas abaixo.

Tipo	**Variável**	**Descrição**
Inteiro	id	Identificador
String	descrição	Descrição da tarefa
Inteiro	status	Status da tarefa (pendente, em progresso, concluída, cancelada)
String	prazo	Prazo da tarefa

Exemplos de estruturas para tarefa e lista de tarefas:

```
typedef struct{
    int id;
    char descricao[50];
    int status;
```

```
typedef struct{
    tarefa *tarefas;
    int qtdTarefas;
}gerenciador;
```

```
    char prazo[11]; //Formato DD-MM-AAAA
}tarefa;
```

4. Gestão de estoque

Os dados abaixo devem ser informados e armazenados sobre cada livro. Deve-se criar um **struct** para representar o livro contendo as variáveis descritas abaixo.

Tipo	Variável	Descrição
Inteiro	id	Identificador
String	descrição	Descrição do produto
Inteiro	categoria	Categoria do produto
Inteiro	quantidade	Quantidade disponível
Real	valor	Valor do produto

Exemplos de estruturas para o produto e lista de produtos:

```
typedef struct{
    int id;
    char descricao[50];
    int categoria;
    float valor;
    int quantidade;
}produto;
```

```
typedef struct{
    produto *produtos;
    int qtdProdutos;
}estoque;
```

Menu sugerido

Opção	Função	Detalhes
1	inserirInicio	Recebe como parâmetros um registro e um ponteiro para uma lista sequencial, retorna o valor 0 ou 1, indicando se o registro foi inserido com sucesso. Todos os registros existentes na lista devem ser deslocados à direita.

2	inserirFim	Recebe como parâmetros um registro e um ponteiro para lista sequencial, retorna o valor 0 ou 1, indicando se o número foi inserido com sucesso.
3	inserirPosição	Recebe como parâmetros um registro, um número inteiro (posição), e um ponteiro para lista sequencial, retorna o valor 0 ou 1, indicando se o registro foi inserido com sucesso. Todos os registros existentes na lista a partir da posição passada como parâmetro devem ser deslocados à direita.
4	listar	Imprime cada elemento da lista por linha, onde cada linha deve apresentar os dados do registro.
5	removerPosição	Recebe como parâmetros um número inteiro(posição) e um ponteiro para uma lista sequencial, retorna o valor 0 ou 1, indicando se o registro foi removido com sucesso. Todos os registros existentes na lista, a partir da posição passada como parâmetro, devem ser deslocados à esquerda.
6	removerValor	Recebe como parâmetros um número inteiro(identificador) e um ponteiro para uma lista sequencial, retorna o valor 0 ou 1, indicando se o número foi removido com sucesso. Caso o valor passado como parâmetro exista, todos os números existentes na lista, a partir da sua posição, devem ser deslocados à esquerda.
7	procurar	Recebe como parâmetros um identificador (id) e retorna a posição na lista em que o valor id se encontra. Caso o id não exista, a função deve retornar o valor -1.
8	tamanho	Recebe como parâmetro o ponteiro para uma lista sequencial e retorna a quantidade de elementos armazenados na lista.
-1	sair	Encerrar a execução e salvar os dados em arquivo

Observação: outras funcionalidades do sistema de gestão devem ser consideradas a depender do problema escolhido. Por exemplo, a função de listagem deve envolver outro menu para listar de acordo com um filtro a ser escolhido. **Note que há uma diferença entre as operações do TAD e as funcionalidades do sistema de gestão.** O TAD deve ser definido no arquivo de cabeçalho .h (listaseq.h, por exemplo) e as funções serão implementadas no .c (listaseq.c, para o exemplo dado) correspondente. O sistema de gestão será implementado em outro arquivo .c (trabalho1.c, por exemplo) que conterá a função main e funções diversas do sistema.