**Estrutura de Dados I – Lista de Exercícios 02**

**P.S.: Códigos das questões que exige programação devem ser incluidos (.c)**

**Entrega no dia (29/05)**

1) Que é um Tipo Abstrato de Dados (TAD) e qual a característica fundamental na sua utilização?

*Um Tipo Abstrato de Dados (TAD) é um modelo que define um conjunto de dados e operações sobre eles, ocultando a implementação interna. Sua principal característica é a abstração: o usuário sabe o que o TAD faz, mas não como ele faz.,*

*ou*

*Um tipo abstrato de dados (TAD) trata-se de um conjunto de dados estruturados e um conjunto de operações que podem ser executadas sobre esses dados. Alguns tipos são muito utilizados em computação: pilha e fila.*

2) Considere dois programas envolvendo um cadastro de funcionários. O programa A foi construído de acordo com os princípios de TAD. Já o programa B não. Diferencie o programa A do programa B.

*O programa* ***A****, que segue os princípios de TAD, possui estrutura modular e encapsula os dados dos funcionários, permitindo acesso apenas por meio de funções específicas, o que facilita manutenção e reaproveitamento.*

*Já o programa* ***B*** *expõe diretamente os dados, tornando o código mais propenso a erros, difícil de manter e menos organizado.*

3) Quais as vantagens de se programar com TADs?

*As principais vantagens de se programar com TADs são:  
- Modularidade, permitindo separar cada parte do programa;  
- Facilidade de manutenção, já que mudanças internas não afetam quem usa o TAD;  
- Reutilização de código, pois o TAD pode ser usado em diferentes programas;  
- Segurança, evitando acesso direto aos dados internos;  
- Organização, deixando o código mais claro e estruturado.*

4) Faça a especificação de um sistema de controle de empréstimos de uma biblioteca usando TAD: diga quais são os dados, as operações, como organizar os dados e operações durante a implementação, etc.

*TAD: Controle de Empréstimos de Biblioteca*

***Dados (informações armazenadas):***

*- Livro: título, autor, código, disponibilidade.*

*- Usuário: nome, matrícula, lista de livros emprestados.*

*- Empréstimo: código do livro, matrícula do usuário, data de retirada, data de devolução.*

***Operações (funções do TAD):***

*- cadastrarLivro()*

*- cadastrarUsuario()*

*- emprestarLivro()*

*- devolverLivro()*

*- verificarDisponibilidade()*

*- listarEmprestimosUsuario()*

***Organização:***

*- Criar TADs separados: Livro, Usuario e Emprestimo.*

*- Cada TAD possui seus dados encapsulados e funções associadas.*

*- O controle de empréstimos manipula os dados via funções, sem acesso direto.*

*Resumo:  
Cada estrutura representa um TAD. O controle de empréstimos se faz por meio de funções bem definidas, garantindo organização, segurança e facilidade de manutenção.*

5) O que é e para que serve uma pilha?

*Uma pilha é uma estrutura de dados do tipo LIFO (Last In, First Out), onde o último elemento inserido é o primeiro a ser removido. Serve para armazenar dados de forma organizada, permitindo operações como empilhar (push) e desempilhar (pop). É útil em situações como desfazer ações, percorrer recursivamente estruturas ou avaliar expressões matemáticas.*

6) O que significa alocação seqüencial de memória para um conjunto de elementos?

*Alocação sequencial de memória significa reservar um bloco contínuo de memória para armazenar todos os elementos de um conjunto, como em vetores. Isso permite acesso rápido por índice, mas exige saber o tamanho previamente e dificulta inserções ou remoções no meio da estrutura.*

7) O que significa alocação estática de memória para um conjunto de elementos?

*Alocação estática de memória significa que o espaço necessário para armazenar os elementos é reservado em tempo de compilação, com tamanho fixo. Ou seja, a quantidade de memória é definida antes da execução do programa e não pode ser alterada durante sua execução.*

8) Faça o esquema de uma implementação seqüencial e estática de uma pilha e descreva seu funcionamento.

#define MAX 100

typedef struct {

int dados[MAX]; // vetor fixo

int topo; // indica a próxima posição livre

} Pilha;

*Funcionamento:*

*- Criação: inicializa topo = 0.*

*- Empilhar (push): insere o elemento em dados[topo] e incrementa topo.*

*- Desempilhar (pop): decrementa topo e retorna dados[topo].*

*- Topo vazio: topo == 0 → pilha vazia.*

*- Pilha cheia: topo == MAX.*

*Essa implementação usa vetor (alocação sequencial) com tamanho fixo (alocação estática).*

9) Desenvolva uma rotina para inverter a posição dos elementos de uma pilha P.

void inverterPilha(Pilha\* p) {

Pilha aux = {.topo = -1};

int temp;

// Transfere todos os elementos de p para a pilha auxiliar

while (p->topo >= 0) {

temp = pop(p);

push(&aux, temp);

}

// Transfere de volta para p (agora invertida)

while (aux.topo >= 0) {

temp = pop(&aux);

push(p, temp);

}

}

10) Desenvolva uma função para testar se uma pilha P1 tem mais elementos que uma pilha P2.

int pilhaMaior(Pilha \*p1, Pilha \*p2) {

return (p1->topo > p2->topo);

}

11) Desenvolva uma função para testar se duas pilhas P1 e P2 são iguais.

int pilhasIguais(Pilha \*p1, Pilha \*p2) {

if (p1->topo != p2->topo) return 0;

for (int i = 0; i < p1->topo; i++) {

if (p1->vet[i] != p2->vet[i]) return 0;

}

return 1;

}

12) O que é e como funciona uma estrutura do tipo fila?

*Uma fila é uma estrutura de dados linear que funciona seguindo o princípio FIFO (First In, First Out), ou seja, o primeiro elemento a entrar é o primeiro a sair. Ela se assemelha a uma fila de pessoas em um banco, onde quem chega primeiro é atendido primeiro. As operações básicas são: enfileirar (adicionar) elementos no final da fila e desenfileirar (remover) elementos do início da fila.*

13) Em que situações uma fila pode ser utilizada?

*Filas são utilizadas em situações onde a ordem de processamento é importante e segue o critério "quem chega primeiro é atendido primeiro". Exemplos incluem:*

*- Sistemas Operacionais: Gerenciamento de tarefas em uma CPU, fila de impressão, buffer de teclado.*

*- Redes de Computadores: Buffers de pacotes em roteadores, gerenciamento de tráfego.*

*- Simulações: Modelagem de filas em bancos, supermercados, call centers.*

*- Estruturas de Dados Internas: Implementação de algoritmos como busca em largura (BFS).*

*- Processamento de Eventos: Enfileiramento de eventos em jogos ou sistemas de interface gráfica.*

14) Faça um esquema da implementação estática e seqüencial de uma fila e e explique resumidamente o funcionamento.

#define MAX 50

typedef struct {

int dados[MAX];

int inicio;

int fim;

} Fila;

void inicializa(Fila \*f) {

f->inicio = f->fim = 0;

}

int enfileira(Fila \*f, int valor) {

if (f->fim == MAX) return 0; // Fila cheia

f->dados[f->fim++] = valor;

return 1;

}

int desenfileira(Fila \*f, int \*valor) {

if (f->inicio == f->fim) return 0; // Fila vazia

\*valor = f->dados[f->inicio++];

return 1;

}

*Nessa implementação sequencial e estática, a fila é representada por um vetor fixo. Os índices inicio e fim controlam as posições onde os elementos são retirados e inseridos. Inserções ocorrem no final e remoções no início da fila.*

15) Desenvolva uma função (com parâmetros) para testar se uma fila F1 tem mais elementos do que uma fila F2 (não se esqueça de mexer nas filas apenas através de seus operadores primitivos.

int filaMaior(Fila \*f1, Fila \*f2) {

int tamanho1 = f1->fim - f1->inicio;

int tamanho2 = f2->fim - f2->inicio;

return (tamanho1 > tamanho2);

}

16) Implemente uma fila em um vetor circular, sem armazenar o número total de elementos (sugestão: nunca deixe que o indicador “fim” alcance o indicador “início”, ainda que seja necessário perder uma posição do vetor.

int cheia(Fila \*f) {

return (f->fim + 1) % MAX == f->inicio;

}

int vazia(Fila \*f) {

return f->inicio == f->fim;

}

int enfileira(Fila \*f, int x) {

if (cheia(f)) return 0;

f->dados[f->fim] = x;

f->fim = (f->fim + 1) % MAX;

return 1;

}

int desenfileira(Fila \*f, int \*x) {

if (vazia(f)) return 0;

\*x = f->dados[f->inicio];

f->inicio = (f->inicio + 1) % MAX;

return 1;

}

17) Implemente a funcionalidade de uma fila a partir de um TAD pilha (sugestão: use 2 pilhas).

void enfileira(Fila \*f, int x) {

push(&f->entrada, x);

}

int desenfileira(Fila \*f) {

int x;

if (vazia(&f->saida)) {

while (!vazia(&f->entrada)) {

push(&f->saida, pop(&f->entrada));

}

}

if (!vazia(&f->saida)) return pop(&f->saida);

return -1;

}

18) Qual a diferença entre alocação seqüencial e alocação encadeada?

*A principal diferença entre alocação sequencial e alocação encadeada reside na forma como os elementos são armazenados na memória e como são acessados.*

***- Alocação Sequencial:*** *Os elementos são armazenados em posições de memória contíguas (uma após a outra). Isso permite acesso direto (indexado) a qualquer elemento, mas dificulta inserções e remoções no meio da estrutura, que podem exigir a movimentação de muitos elementos. Exemplos incluem arrays e vetores.*

***- Alocação Encadeada:*** *Os elementos são armazenados em posições de memória não contíguas. Cada elemento (ou nó) contém o dado e um ponteiro para o próximo elemento na sequência. Isso facilita inserções e remoções (basta ajustar os ponteiros), mas exige acesso sequencial (percorrendo os ponteiros a partir do início), e há um custo adicional de memória para armazenar os ponteiros. Exemplos incluem listas encadeadas.*

19) Implemente o TAD da lista encadeada e dinâmica.

typedef struct no {

int valor;

struct no \*prox;

} No;

No\* insere(No \*lista, int x) {

No \*novo = malloc(sizeof(No));

novo->valor = x;

novo->prox = lista;

return novo;

}

void imprime(No \*lista) {

while (lista) {

printf("%d ", lista->valor);

lista = lista->prox;

}

}

20) Utilizando o TAD lista anterior, modele e implemente um sistema de cadastro de alunos de uma universidade.

typedef struct item {

char nome[30];

struct item \*prox;

} Item;

Item\* inserirItem(Item \*lista, char nome[]) {

Item \*novo = malloc(sizeof(Item));

strcpy(novo->nome, nome);

novo->prox = lista;

return novo;

}

Item\* removerItem(Item \*lista, char nome[]) {

Item \*ant = NULL, \*atual = lista;

while (atual && strcmp(atual->nome, nome) != 0) {

ant = atual;

atual = atual->prox;

}

if (!atual) return lista;

if (!ant) lista = atual->prox;

else ant->prox = atual->prox;

free(atual);

return lista;

}

21) Utilizando o TAD lista anterior, modele e implemente um sistema de lista de compras para uma casa: cada item de cozinha que acaba é inserido na lista para ser comprado na próxima ida ao mercado; quando um item é comprado, ele deve sair da lista.

typedef struct item {

char nome[30];

struct item \*prox;

} Item;

Item\* inserirItem(Item \*lista, char nome[]) {

Item \*novo = malloc(sizeof(Item));

strcpy(novo->nome, nome);

novo->prox = lista;

return novo;

}

Item\* removerItem(Item \*lista, char nome[]) {

Item \*ant = NULL, \*atual = lista;

while (atual && strcmp(atual->nome, nome) != 0) {

ant = atual;

atual = atual->prox;

}

if (!atual) return lista;

if (!ant) lista = atual->prox;

else ant->prox = atual->prox;

free(atual);

return lista;

}

22) Quais as diferenças entre os dois sistemas anteriores? Quais foram as vantagens e desvantagens de se usar TAD?

***Diferenças:***

*- O sistema de alunos usa uma lista com dados estruturados (nome + matrícula), enquanto a lista de compras lida com nomes simples.*

*- O cadastro de alunos normalmente exige busca por matrícula, já a lista de compras prioriza inserção e remoção por nome.*

***Vantagens de usar TAD:***

*- Organização do código.*

*- Reutilização das funções.*

*- Encapsulamento dos dados (evita manipulação direta).*

***Desvantagens:***

*- Pode ser mais trabalhoso para programas pequenos.*

*- Aumenta a complexidade inicial para iniciantes.*

23) Usando os conceitos de TAD, implemente uma pilha sobre uma lista.

typedef struct no {

int valor;

struct no \*prox;

} No;

typedef struct {

No \*topo;

} Pilha;

void push(Pilha \*p, int x) {

No \*novo = malloc(sizeof(No));

novo->valor = x;

novo->prox = p->topo;

p->topo = novo;

}

int pop(Pilha \*p) {

if (!p->topo) return -1;

No \*temp = p->topo;

int v = temp->valor;

p->topo = temp->prox;

free(temp);

return v;

}

24) Usando os conceitos de TAD, implemente uma fila sobre uma lista.

typedef struct no {

int valor;

struct no \*prox;

} No;

typedef struct {

No \*ini, \*fim;

} Fila;

void enfileira(Fila \*f, int x) {

No \*novo = malloc(sizeof(No));

novo->valor = x;

novo->prox = NULL;

if (f->fim) f->fim->prox = novo;

else f->ini = novo;

f->fim = novo;

}

int desenfileira(Fila \*f) {

if (!f->ini) return -1;

No \*temp = f->ini;

int v = temp->valor;

f->ini = temp->prox;

if (!f->ini) f->fim = NULL;

free(temp);

return v;

}

25) Por que se pode modelar uma pilha sobre uma lista mas não se pode manipular uma pilha como uma fila?

*Porque pilhas (LIFO) operam no topo, como listas encadeadas no início, permitindo empilhar e desempilhar com eficiência. Já filas são FIFO e exigem controle de início e fim. Usar apenas uma pilha não mantém a ordem de uma fila. Para simular fila com pilhas, seriam necessárias* ***duas*** *para inverter a ordem dos elementos.*

26) O que é uma lista linear? E uma lista não linear? Dê exemplos e justifique essa nomenclatura (linear vs. não linear).

***Lista Linear:*** *Elementos organizados em uma* ***sequência contínua****, um após o outro, com um único predecessor e sucessor. Exemplos: arrays, listas encadeadas, pilhas e filas. Justificativa: Pode-se percorrer a lista em uma única direção, como uma linha.*

***Lista Não Linear:*** *Elementos organizados de forma* ***não sequencial****, podendo ter múltiplos predecessores ou sucessores, formando relações hierárquicas ou de rede. Exemplos: árvores (nós com múltiplos filhos) e grafos (nós com múltiplas conexões). Justificativa: As relações não seguem uma única linha, permitindo múltiplos caminhos.*

27) Utilizando as operações de manipulação de pilhas vistas em sala, uma pilha auxiliar e uma variável do tipo TipoItem, escreva um procedimento que remove um item com chave c de uma posição qualquer de uma pilha. Note que você não tem acesso à estrutura interna da pilha (topo, item, etc), apenas às operações de manipulação.

void removeItem(Pilha \*p, int c) {

Pilha aux;

inicializa(&aux);

TipoItem x;

// Desempilha tudo até encontrar o item

while (!vazia(p)) {

x = desempilha(p);

if (x != c)

empilha(&aux, x);

else

break; // achou o item, não empilha de volta

}

// Devolve os elementos para a pilha original

while (!vazia(&aux)) {

empilha(p, desempilha(&aux));

}

}

28) Considero que um estacionamento da Rua Direita, em Trindade, é composto por uma única alameda que guarda até dez carros. Existe apenas uma entrada/saída no estacionamento, e esta extremidade da alameda dá acesso justamente à Rua Direita. Se chegar um cliente para retirar um carro que não seja o mais próximo da saída, todos os carros bloqueando seu caminho sairão do estacionamento. O carro do cliente será manobrado para fora do estacionamento, e os outros carros voltarão a ocupar a mesma sequência inicial. Escreva um programa que processe um grupo de linhas de entrada. Cada linha de entrada contém um ‘E’, de entrada, ou um ‘S’ de saída, e o número da placa do carro. Presume-se que os carros cheguem e partam na mesma ordem que entraram no estacionamento. O programa deve imprimir uma mensagem sempre que um carro chegar ou sair. Quando um carro chegar, a mensagem deve especificar se existe ou não vaga para o carro no estacionamento. Se não houver vaga, o carro partirá sem entrar no estacionamento. Quando um carro sair do estacionamento, a mensagem deverá incluir o número de vezes em que o carro foi manobrado para fora do estacionamento para permitir que os outros carros saíssem.

29) Considere uma pilha P vazia e uma fila F não vazia. Utilizando apenas os testes de fila e pilha vazias, as operações Enfileira, Desenfileira, Empilha, Desempilha, e uma variável aux do TipoItem, escreva uma função que inverta a ordem dos elementos da fila.

void inverterFila(Fila \*f, Pilha \*p) {

TipoItem aux;

while (!filaVazia(f)) {

aux = desenfileira(f);

empilha(p, aux);

}

while (!pilhaVazia(p)) {

aux = desempilha(p);

enfileira(f, aux);

}

}

30) Considere listas implementadas por arranjos, então pede-se para implementar funções que:

• Localize/ Pesquise/Encontre (search) elementos

• Concatenar/intercalar (Merge) duas listas

• Dividir uma lista em várias (k)

• Copiar uma lista

• Ordenar (sort) ima lista por ordem crescente/decrescente

31) Escreva uma função em C para trocar os elementos m e n de uma lista (m e n podem ser chaves ou mesmo ponteiros para os elementos – a escolha é sua).

#include <stdio.h>

void troca(int \*v, int m, int n) {

int temp = v[m];

v[m] = v[n];

v[n] = temp;

}

int main() {

int v[] = {1, 2, 3, 4, 5};

int m = 1, n = 3; // índices para trocar (2 e 4)

troca(v, m, n);

for (int i = 0; i < 5; i++) printf("%d ", v[i]);

return 0;

}

32) Se você tem de escolher entre uma representação por lista encadeada ou uma representação usando posições contíguas de memória para um vetor, quais informações são necessárias para você selecionar uma representação apropriada? Como esses fatores influenciam na escolha da representação?

***1 Tamanho fixo ou variável dos dados:***

*Se o número de elementos é conhecido e fixo → vetor é mais eficiente.*

*Se o tamanho muda com frequência → lista encadeada é mais flexível.*

***2 Frequência de inserções e remoções:***

*Se ocorrem muitas inserções/remoções no meio → lista encadeada é melhor.*

*Se os acessos são principalmente por índice → vetor é mais rápido.*

***3 Uso de memória:***

*Vetores ocupam espaço fixo, podendo desperdiçar memória.*

*Listas usam memória sob demanda, mas com sobrecarga por ponteiros.*

*- Influência:*

*Esses fatores afetam o desempenho, o consumo de memória e a facilidade de manutenção. A escolha ideal depende do comportamento esperado da aplicação.*