Exercicios Lista, Pilhas e Filas

Aluno: William Cardoso Barbosa+

1. Qual a diferença entre Tipo de Dado, Tipo Abstrato de Dado e Estrutura de Dado?O que é um tipo abstrato de dados (TAD) ?
    a. Tipo de dado é a forma que determinado valor pode assumir , po exemplo, float, inteiro ,char e struct.
    b. Tipo abstrato de dado(TAD) é o pensamento de como os dados serão manipulados, o conceito por trás da implementação. Logo, ele se preocupa em definir o comportamento da estrutura de dados.
    c. Estrutura de dados é a forma como os dados são organizados , ou seja, o relacionamento lógico do tipo de dado.
2. Quais são as vantagens de se usar TAD?
    a. Modularização da aplicação, legibilidade e ecapsulamento, pois separa o conceito da aplicação de sua implementação.
3. Como especificar um TAD?
    a. Um TAD deve conter quais são os tipos de dados, a abstração da estrutura de dados e as funcionalidades que podem ser usadas com essa estrutura de dados. Essa primeira ideia fica no conceito da abstração, a segunda etapa seria a de implementação, separada do conceito, a implementação consumiria as informações fornecidas pelo conceito e trabalharia em cima disso.

4. Quais são as vantagens e as desvantagens de se usar Lista Sequencial Estática? Quando devemos usá-la?
    a. Desvantagens
        i. deslocamento dos itens ao remover ou adicionar, dependendo da posição.
        ii. Número definido de itens que devem compor a lista.
        iii. Seus espaço é pré-definido na memória.
    b. vantagens
        i. Acesso direto a um elemento pelo indice.
        ii. ordenação
        iii. fácil implementação
    c. Quando usá-las:
        i. Quando o problema tiver um tamanho bem definido.
        ii. Quando a remoção e inserção não for em diferentes partes da lista , apenas no final. Isso faz com que não haja deslocamento de itens.
        iii. quantidade pequena de itens.

5. Crie o Tipo Abstrato de Dados (TAD) lista linear ordenada (pelo campo chave), em C, de forma modular. Considere sua implementação sequencial e encadeada dinamicamente, e inclua procedimentos para:
   a. Verificar se uma lista esta ordenada ou não (a ordem pode ser crescente ou decrescente)
   b. Fazer uma cópia da lista L1 em uma outra lista L2;
   c. Fazer uma cópia da lista L1 em outra L2, eliminando os elementos repetidos;
   d. Inverter uma lista L1 colocando o resultado em L2;
   e. Intercalar duas listas, L1 e L2, gerando uma lista L3. Considere que L1, L2 e L3 estão ordenadas;
   f. Dada uma lista L1, gerar uma lista L2 onde cada registro cont em dois campos de informacao: elem, que contem um elemento de L1 e count, que contem quantas vezes este elemento apareceu em L1;
   g. Assumindo que os elementos de uma lista L1 sao inteiros positivos, fornecer os elementos que aparecem o maior e o menor número de vezes (forne¸ca os elementos e o n´umero de vezes correspondente).

```
// lista.h
#define MAX 100

typedef struct  {
  int chave;
}ITipoElem;


typedef struct {
  ITipoElem A[MAX];
  int Nelem;
}ILista;


typedef struct {
  int chave;
  int cont;
}ITipoElemCont;

typedef struct {
  ITipoElemCont A[MAX];
  int Nelem;
}IListaReg;

int iniciar(ILista *L);

void exibir_lista(ILista *L);

void exibir_lista_registro(IListaReg *L);

int aux_vazia(ILista *L);

int ordenada(ILista *L);

void copia_lista(ILista *L, ILista *P);

void copia_lista_sem_repetidos(ILista *L, ILista *P);

void inverte_lista(ILista *L, ILista *P);

void intercala_lista(ILista *L, ILista *P, ILista *K);
```

```
void contador_registros(ILista *L, IListaReg *K);

void frequencia_lista(ILista *L);
```

```
//lista.cpp

#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

int iniciar(ILista *L){
  L ->Nelem = 5;
  return 1;
}

int aux_vazia(ILista *L) {
  return L->Nelem == 0;
}

void exibir_lista(ILista *L){
  int i;
  if(aux_vazia(L)) printf("Nao ha elementos na lista");
  for(i=0;i<L->Nelem;i++){
    printf("%d ", L->A[i].chave);
  }
}

void exibir_lista_registro(IListaReg *L){
  int i;
  for(i=0;i<L->Nelem;i++){
    printf("%d %d\n", L->A[i].chave, L->A[i].cont);
    printf("\n");
  }
}

int ordenada(ILista *L){
  if(aux_vazia(L)) return 0;
  int nCrescente = 0;
  int nDecrescente = 0;
  for(int i  = 0; i < (L->Nelem) - 1; i++){
    if(L->A[i].chave < L->A[i +1].chave) nCrescente++;
    if(L->A[i].chave > L->A[i +1].chave) nDecrescente++;
  }
  if((nDecrescente == L->Nelem - 1) || (nCrescente == L->Nelem -1)) return 1;
  return 0;
}

void copia_lista(ILista *L, ILista *P){
  if(aux_vazia(L))  printf("\na primeira lista nao pode ser vazia.");
  int i = 0;
  P->Nelem = 0;
  while(i < L->Nelem) {
    P->A[i] = L->A[i];
    P->Nelem++;
    i++;
  }
}

void copia_lista_sem_repetidos(ILista *L, ILista *P){
  int i = 0,j = 0;
  while(i < L->Nelem){
    while(j < P->Nelem){
      if(L->A[i].chave == P->A[j].chave) break;
      j++;
    }
    if(j == P->Nelem){
      P->A[P->Nelem] = L->A[i];
```

```
      P->Nelem++;
    }
    i++;
  }
}

void inverte_lista(ILista *L, ILista *P){
  if(aux_vazia(L)) printf("\na primeira lista nao pode ser vazia.");
  P->Nelem = 0;
  int i, j;
  for( i  =  L->Nelem, j = 0; i >= 0; i--, j++){
    P->A[j] = L->A[i];
  }
  P->Nelem = j;
}

void intercala_lista(ILista *L, ILista *P, ILista *K) {

  if(aux_vazia(L) || aux_vazia(P)) printf("\na primeira e a segunda lista nao podem ser vazia.");
  int i = 0, j = 0, k = 0;
  while(i < L->Nelem && j < P->Nelem){
    if(L->A[i].chave < P->A[j].chave){
      K->A[k] = L->A[i];
      i++;
    }
    else{
      K->A[k] = P->A[j];
      j++;
    }
    k++;
  }
  while(i < L->Nelem){
    K->A[k] = L->A[i];
    i++;
    k++;
  }
  while(j < P->Nelem){
    K->A[k] = P->A[j];
    j++;
    k++;
  }
  K->Nelem = k;
}

void contador_registros(ILista *L, IListaReg *K){
  if(aux_vazia(L)) printf("\na lista nao pode ser vazia.");
  int i, j;
  for(i = 0; i<L->Nelem; i++){
    K->A[i].chave = L->A[i].chave;
    K->A[i].cont = 0;
    for(j = 0; j <L->Nelem; j++){
      if(L->A[i].chave == L->A[j].chave){
        K->A[i].cont++;
      }
    }
  }
  K->Nelem = L->Nelem;
}

void frequencia_lista(ILista *L) {
  int i , j;
  IListaReg K;
  contador_registros(L, &K);
  int indice_maior, maior, indice_menor, menor;
  for(i = 0; i<(K).Nelem; i++) {
    if(i == 0){
      indice_maior = 0;
      indice_menor =  0;
      menor = (K).A[i].cont;
      maior = (K).A[i].cont;
```

```
    }
    if((K).A[i].cont > maior){
      maior =  (K).A[i].cont;
      indice_maior = i;
    }

    if((K).A[i].cont < menor){
      menor =  (K).A[i].cont;
      indice_menor = i;
    }
  }
  printf("\nO maior registro e sua frequencia : %d %d", (K).A[indice_maior].chave, (K).A[indice_maior].cont);
  printf("\nO menor registro e sua frequencia : %d %d", (K).A[indice_menor].chave, (K).A[indice_menor].cont);
}
```

6. Considere as seguintes declara¸c˜oes:

```
#define n 100
int a[n];
int ult;
```

Considere tambem que no array acima esta sequencialmente armazenada uma lista ordenada, cujo ultimo elemento e apontado por ult. Declare, em C, as seguintes operacoes:
(a) insere (v, a) - dado o inteiro v, ele deve inseri-lo na lista a, caso ele ja nao esteja la. Se ja estiver, nada faz.
(b) elimina (v, a) - elimina o registro com valor v da lista a, caso ele esteja la. Se nao estiver, imprime uma mensagem.
(c) busca (v, a) - verifica se o valor v pertence a lista, retornando o valor do ındice do array onde ele esta, neste caso, ou 0, caso contrario. Fa¸ca duas versoes: uma com sentinela e outra sem sentinela.

```
//lista.h

#define MAX 100

typedef struct {
  int chave;
}tipo_elem;

typedef struct {
  tipo_elem A[MAX];
  int ult; // SE ULT APONTA PARA O ULTIMO ELEMENTO, ENTAO ELE É A QUANTIDADE DE ELEMENTOS, POSTO QUE ESTÁ EM INTEIRO.
}Lista;



void inserir(Lista *L, tipo_elem v);
int cheia(Lista *L);
int vazia(Lista *L);
void iniciar(Lista *L);
int verificar_item(Lista *L, tipo_elem v);
void elimina(Lista *L, tipo_elem v);
int busca(Lista *L, tipo_elem v);
```

```
//lista.cpp
#include <stdlib.h>
#include <stdio.h>
#include "lista.h"

void inserir(Lista *L, tipo_elem v){
  if(cheia(L)) printf("\nlista cheia");
  int num_existe = verificar_item(L, v);
  if(num_existe != -1) printf("\nnumero ja esta na lista.");
  else {
    L->ult++;
    L->A[L->ult] = v;
  }
}
void iniciar(Lista *L){
  L->ult = 0;
}
int cheia(Lista *L) {
 return L->ult == MAX;
}
int vazia(Lista *L){
  return L->ult == 0;
}
int verificar_item(Lista *L, tipo_elem v){
  int i;
  for(i = 0; i <= L->ult; i++){
    if(L->A[i].chave == v.chave) return i;
  }
  return -1;
}

int verificar_item_sentinela(Lista *L, tipo_elem v){
  int i;
  int achou = 0;
  for(i = 0; i < L->ult; i++) {
    if(L->A[i].chave == v.chave){
      achou = 1;
      break;
    }
  }
  if(achou) return i;
  return -1;

}
void elimina(Lista *L, tipo_elem v){
  if(vazia(L)) printf("\nlista vazia");
  int num_existe = verificar_item(L, v);
  if(num_existe != -1){
    int i;
    for(i = num_existe; i < L->ult; i++) {
      L->A[i] = L->A[i+1];
    }
    L->ult--;
  } else {
    printf("\nnumero nao esta na lista.");
  }
}
int busca(Lista *L, tipo_elem v){

}
```

7. Seja L = (a1, a2, a3, ..., an) uma lista linear representada num vetor V [n]. Usando o mapeamento o i-´esimo elemento de L ´e armazenado em V [i − 1]. Escreva um algoritmo

para reverter a ordem dos elementos em V, isto ´e, o algoritmo deve transformar V tal que V[i] contenha o elemento n - i de L. O ´unico espa¸co adicional dispon´ıvel para seu algoritmo ´e suficiente para apenas uma vari´avel simples. A entrada para seu algoritmo ´e V e n.

a.

```
void reverte_ordem_elem(int *V, int n){
  int i, j, aux;
  for(i = 0, j = n - 1; i < j; i++, j--){
    aux = V[i];
    V[i] = V[j];
    V[j] = aux;
  }
}

main() {
  int V[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  reverte_ordem_elem(V, 10);
  int i;
  for(i = 0; i < 10; i++) {
    printf("%d ", V[i]);
  }
  printf("\n");

}
```

8. Fa¸ca um algoritmo que inverta uma lista encadeada, isto ´e, o ´ultimo elemento passa a ser o primeiro, o pen´ultimo passa a ser o segundo, e assim por diante, e o primeiro passa a ser o ´ultimo. Fa¸ca a invers˜ao atrav´es da invers˜ao dos campos de liga¸c˜ao, e NAO dos campos ˜ de informa¸c˜ao

a.

```
//lista.h
#define MAX  100
#define NIL -1


typedef struct {
  int chave;
}Tipo_elem;

typedef struct {
  Tipo_elem info;
  int link;
}No;

typedef struct  {
  No V[MAX];
  int inicio, dispo;
}Lista;

void iniciar(Lista *L);
void inserir(Lista *L, Tipo_elem v);
void inverte_lista(Lista *L);
void exibir(Lista *L);
```

b.

```
//lista.cpp
#include <stdlib.h>
#include <stdio.h>
#include "lista.h"

/*
Fa¸ca um algoritmo que inverta uma lista encadeada, isto ´e, o ´ultimo elemento passa a ser
o primeiro, o pen´ultimo passa a ser o segundo, e assim por diante, e o primeiro passa a ser
o ´ultimo. Fa¸ca a invers˜ao atrav´es da invers˜ao dos campos de liga¸c˜ao, e NAO dos campos ˜
de informa¸c˜ao
*/


void iniciar(Lista *L){
  L->inicio = NIL;
  L->dispo = 0;
  int i;
  for(i = 0; i < MAX; i++) {
    L->V[i].link = i + 1;
  }
  L->V[MAX - 1].link = NIL;
}
void inserir(Lista *L, Tipo_elem v){
  int i = L->dispo;
  L->V[i].info = v;
  L->V[i].link = L->inicio;
  L->inicio = i;
  L->dispo = L->V[i].link;
}

void inverte_lista(Lista *L){
  int i, j, k;
  i = L->inicio;
  j = L->V[i].link;
  L->V[i].link = NIL;
  while(j != NIL) {
    k = j;
    j = L->V[j].link;
    L->V[k].link = L->inicio;
    L->inicio = k;
  }
}

void exibir(Lista *L){
  int i;
  for(i = L->inicio; i != NIL; i = L->V[i].link){
    printf("%d->%d ", L->V[i].info.chave, L->V[i].link);
  }
  printf("\n");
}

main() {

}
```

9. Dada uma lista encadeada que armazena n´umeros inteiros escreva uma fun¸c˜ao que transforma a lista dada em duas listas encadeadas: a primeira contendo os elementos cujo conte´udo ´e par e a segunda contendo os elementos com conte´udos ´ımpares. A fun¸c˜ao n˜ao deve manipular o conte´udo dos n´os, ou seja, n˜ao deve copiar o conte´udo de lado pra outro; a fun¸c˜ao deve manipular apenas os ponteiros.

```
void inserir(Lista *L, Tipo_elem v){
  int i = L->dispo;
  L->V[i].info = v;
  L->V[i].link = L->inicio;
  L->inicio = i;
  L->dispo = L->V[i].link;
}

void par_impar(Lista *L){
  Lista L1, L2;
  iniciar(&L1);
  iniciar(&L2);
  int i = L->inicio;
  for(i ; i != NIL; i = L->V[i].link) {
    if(L->V[i].info.chave % 2 == 0) {
      inserir(&L1, L->V[i].info);
    } else {
      inserir(&L2, L->V[i].info);
    }
  }
}
```

10. Descreva as vantagens e desvantagens em se usar Listas Est´aticas (Sequenciais), Listas Dinˆamicas Simplesmente Encadeadas, Listas Dinâamicas Duplamente Encadeadas.
    a. vantages
        i. lista estática sequencial
            1. Listas pequenas
            2. quando for inserir e remover elementos do final
            3. busca por indexação.
        ii. Dinâmicas simples encadeadas
            1. A quantidade de informação não é pré-definida
            2. não há deslocamento de itens na inserção e remoção
    b. Desvantagens
        i. Lista estática sequencial
            1. Deslocamento de itens na remoção e inserção de elementos no meio da lista.
            2. A quantidade de itens é pré-definida na mémoria.
        ii. Lista Dinâmica simples encadeada
            1. não tem acesso direto aos elementos
            2. só é possivel fazer busca sequencial.
11. Explique o que é o tipo abstrato de dado Fila.
    a. É uma estrutura que permite o gerenciamento de itens pela adição de elementos no final e a remoção desses no inicio. Dessa forma, as operações ocorrem nas extremidades dessa estrutura, o conceito lógica está em que o primeiro elemento que entra é o primeiro elemento que sai.

12. Crie TADs Fila Estatica Sequencial e Fila Dinamica. Os TADs devem conter pelo menos ¨
as seguintes opera¸c˜oes:
i. Criar (F) - criar uma fila F vazia
ii. Inserir (x, F) - insere x no fim de F
iii. Vazia (F) - testa se F est´a vazia
iv. Primeiro (F) - acessa o elemento do in´ıcio da fila
v. Elimina (F) - elimina o elemento do in´ıcio da fila

```
FilaEstatica.h
#define MAX 100

typedef struct{
  int chave;
}tipo_elem;

typedef  struct {
  tipo_elem A[MAX];
  int Nelem;
}Lista;

void criar(Lista *L);
void inserir (Lista *L, tipo_elem v);
int vazia(Lista *L);
int cheia(Lista *L);
int primeiro(Lista *L);
void remover(Lista *L);
```

```
//filaEstatica.cpp
#include <stdlib.h>
#include <stdio.h>
#include "lista.h"

void criar(Lista *L){
  L->Nelem = 0;
}
void inserir(Lista *L, tipo_elem v){
  if(cheia(L)) printf("\nfila cheia");
  L->A[L->Nelem++] = v;
}
int vazia(Lista *L){
  return L->Nelem == 0;
}
int cheia(Lista *L){
  return L->Nelem == MAX;
}
int primeiro(Lista *L){
  if(vazia(L)) return NULL;
  return L->A[0].chave;
}
void remover(Lista *L){
  if(vazia(L)) printf("\nfila vazia");
  int i;
  for(i = 0; i < L->Nelem; i++) L->A[i] = L->A[i +1];
  L->Nelem--;
}
```

```
//dinamica.h
typedef struct{
  int chave;
} tipo_elem;
```

```
typedef struct No{
  tipo_elem info;
  struct No *prox;
} No;

typedef struct {
  No* Inicio;
} Lista;

void criar(Lista *L);
void inserir (Lista *L, tipo_elem v);
int vazia(Lista *L);
int primeiro(Lista *L);
void remover_final(Lista *L);
void reverso(Lista *L);
int numElem(Lista *L);
```

```
//dinamica.cpp
#include <stdlib.h>
#include <stdio.h>
#include "dinamica.h"


void criar(Lista *L){
  L->Inicio = NULL;

}
int vazia(Lista *L){
  return L->Inicio == NULL;
}
int primeiro(Lista *L){
  if(vazia(L)){
    printf("Lista vazia\n");
    return 0;
  }
  return L->Inicio->info.chave;
}
void remover_final(Lista *L){
  if(vazia(L)){
    printf("Lista vazia\n");
    return;
  }
  No *p;
  p = L->Inicio;
  while(p->prox != NULL){
    p = p->prox;
  }
  free(p);
}
void reverso(Lista *L){
  if(vazia(L)){
    printf("Lista vazia\n");
    return;
  }
  No *p, *q;
  p = L->Inicio;
  q = p->prox;
  p->prox = NULL;
  while(q != NULL){
    No *r = q->prox;
    q->prox = p;
    p = q;
    q = r;
  }
  L->Inicio = p;
}
int numElem(Lista *L){
  if(vazia(L)){
```

```
    printf("Lista vazia\n");
    return 0;
  }
  No *p;
  int cont = 0;
  p = L->Inicio;
  while(p != NULL){
    cont++;
    p = p->prox;
  }
  return cont;
}
```

13. Defina a opera¸c˜ao reverso, que reposiciona os elementos na fila de tal forma que o in´ıcio da fila torna-se o fim, e vice-versa. A fila est´a alocada num array A[M] e ´e vista como um anel. Escreva um procedimento que determine o n´umero de elementos da fila.

```
//lista.cpp
void reverso(Lista *L){
  int i, j;
  for(i = 0, j = L->Nelem - 1; i < j; i++, j--){
    tipo_elem aux = L->A[i];
    L->A[i] = L->A[j];
    L->A[j] = aux;
  }
}
```

14. Escreva um procedimento que determine o n´umero de elementos da fila.

```
int numElem(Lista *L) {
  return L->Nelem;
}
```

15. Dada uma fila com 10 posicoes (que armazenam inteiros) temos o seguinte: enqueue 3, enqueue 4, enqueue 20, enqueue 1, dequeue, dequeue, enqueue 20, enqueue 30, enqueue 60 Use uma tabela para simular o que est´a ocorrendo com a fila.

    a.

```
// []
//1-enqueue  3
//2-enqueue 4
//3-enqueue 20
//4-enqueue 1
//5-dequeue
//6-dequeue
//7-enqueue 20
//8-enqueue 30
//9-enqueue 60

//1-[3]
//2-[3,4]
//3-[3,4,20]
//4-[3,4,20,1]
```

```
//5-[4,20,1]
//6-[20,1]
//7-[20, 1, 20] -> como não restrição de valores repetidos, o 20 foi adicionado ao final
//8-[20,1,20,30]
//9-[20,1,20,30,60]
```

16. Crie TADs Pilha Estatica Sequencial e Pilha Dinamica na linguagem C de forma modular.
    Os TADs devem conter pelo menos as seguintes operacoes:
    i. Criar (P) - criar uma pilha P vazia
    ii. Inserir (x, P) - insere x no topo de P (empilha): push(x,P)
    iii. Vazia (P) - testa se P est´a vazia
    iv. Topo (P) - acessa o elemento do topo da pilha (sem eliminar)
    v. Elimina (P) - elimina o elemento do topo de P (desempilha): pop(P)
    vi. Destruir(P) - destr´oi a estrutura e libera o espa¸co ocupado por ela.

```
//estatica.h
#define MAX 100

typedef struct{
  int chave;
}tipo_elem;

typedef  struct {
  tipo_elem A[MAX];
  int Nelem;
}Lista;

void criar(Lista *L);
void push(Lista *L, tipo_elem v);
int vazia(Lista *L);
int topo(Lista *L);
void pop(Lista *L);
void destruir(Lista *L);
```

```
//estatica.cpp
#include "estatica.h"
#include <stdlib.h>
#include <stdio.h>

void criar(Lista *L){
  L->Nelem = 0;
}
void push(Lista *L, tipo_elem v){
  L->A[L->Nelem] = v;
  L->Nelem++;
}
int vazia(Lista *L){
  return L->Nelem == 0;
}
int topo(Lista *L){
  if(vazia(L)){
    printf("Lista vazia\n");
    return 0;
  }
  return L->A[L->Nelem-1].chave;
}
void pop(Lista *L){
  if(vazia(L)){
    printf("Lista vazia\n");
    return;
```

```
  }
  L->Nelem--;
}
void destruir(Lista *L){
  L->Nelem = 0;
}
```

17. Suponha que haja 4 registros – 1 2 3 4 – nesta ordem, para serem inseridos numa pilha. Qual seria a sequencia correta de operaçoes de inserçao (I) e eliminaçao (E) para se obter os registros na ordem 2 4 3 1? Por exemplo, aplicando-se a sequencia IIEIEE sobre a ordem inicial 1 2 3, obtem-se 2 3 1.

```
//PILHA -> LIFO -> Last in Las out(o último a entrar é o último a sair)
// 1 2 3 4  devem ser inseridos em uma pilha
// quero obter os itens na ordem 2 4 3 1
// I I E I I E E E
```

18. Uma palavra e um palindromo se a sequencia de letras que a forma e a mesma seja ela lida da esquerda para a direita ou da direita para esquerda. Exemplos: arara, raiar, hanah. Escreva a funçao pal´ındromo de maneira que, dada uma palavra, retorne true caso a palavra seja uma pal´ındromo, e false caso contr´ario. Utilize para isso a estrutura de dados pilha.

```
Lista reverse(Lista *L){
  int i, j;
  Lista aux;
  criar(&aux);
  for(i=L->Nelem-1; i>=0; i--){
    push(&aux, L->A[i]);
  }
  return aux;
}

int palindromo(Lista *L){
  Lista aux = reverse(L);
  int i;
  for(i=0; i<L->Nelem; i++){
    if(L->A[i].chave != aux.A[i].chave){
      return 0;
    }
  }
  return 1;
}
```

19. Passe as expressoes aritmeticas abaixo para as notacoes prefixa e posfixa.
(a) A + B * (C + D) / E - B - D

```
// INFIXA: A + B * (C + D) / E - B - D
/*
PREFIXA: A + B * (C + D) / E - B - D
  A + B * [+CD] / E - B -D
  A + [*B+CD] / E - B - D
  A + [/*B+CDE] - B - D
  [+A/*B+CDE] - B - D
```

```
  [-+A/*B+CDEB]  - D
  [--+A/*B+CDEBD]

 POSFIXA: A + B * (C + D) / E - B - D
    A + B * [CD+] / E - B - D
    A +  [BCD+*] / E - B - D
    A +  [BCD+*E/] - B - D
    [ABCD+*E/+] - B - D
    [ABCD+*E/+B-] - D
    [ABCD+*E/+B-D-]



*/
```

(b) (A + B) * D + E / (F + A * D) + C

```
//INFIXA
/*
PREFIXA:(A + B) * D + E / (F + A * D) + C
  [+AB] * D + E / (F + A * D) + C
  [+AB] * D + E / (F + [*AD]) + C
  [+AB] * D + E / [+F*AD] + C
  [+AB] * D + E / [+F*AD] + C
  [*+ABD] + E / [+F*AD] + C
  [*+ABD] + [/E+F*AD] + C
  [+*+ABD/E+F*AD] + C
  [++*+ABD/E+F*ADC]


POSFIXA:  (A + B) * D + E / (F + A * D) + C
  [AB+] * D + E / (F + A * D) + C
  [AB+] * D + E / (F + [AD*]) + C
  [AB+] * D + E / [FAD*+] + C
  [AB+D*] + E / [FAD*+] + C
  [AB+D*] + [EFAD*+/] + C
  [AB+D*EFAD*+/+] + C
  [AB+D*EFAD*+/+C+]
*/
```

20. Escreva um algoritmo para transformar uma expressao em notacao prefixa para a nota¸c˜ao posfixa
21. Dada uma pilha com 10 posicoes (que armazenam inteiros), temos o seguinte: empilha 11, empilha 4, empilha 20, empilha 14, desempilha, desempilha, empilha 10, empilha 20, desempilha Use uma tabela ara simular o que esta ocorrendo com a pilha.

    a.

```
//Pilha
//[]
//1-empilha 11
//2-empilha 4
//3-empilha 20
//4-empilha 14
//5-desempilha
//6-desempilha
//7-empilha 10
//8-empilha 20
//9-desempilha

//1-[11]
```

```
//2-[4,11]
//3-[20,4,11]
//4-[14,20,4,11]
//5-[20,4,11]
//6-[4,11]
//7-[10,4,11]
//8-[20,10,4,11]
//9-[10,4,11]
```

22. Indique qual a melhor estrutura de dados para modelar cada caso a seguir (pilha, fila, fila circular, fila com prioridade, etc.):

a) atendimento de caixa de banco

Fila

b) retirada e colocaçao de caixas (uma sobre a outra) em um estoque em um dep´osito

Pilha

c) lanchonete

Fila

d) atendimento em banco com uma fila preferencial

Fila com prioridade

e) a retirada de pratos empilhados em um armario

Pilha