P1 – ESTRUTURA DE DADOS

Implemente uma fila dinâmica em um programa que simule o controle de uma pista de decolagem de aviões em um aeroporto.

Neste programa, o usuário deve ser capaz de realizar as seguintes tarefas:

a) Listar o número de aviões aguardando na fila de decolagem;

b) Autorizar a decolagem do primeiro avião da fila;

c) Adicionar um avião à fila de espera;

d) Listar todos os aviões na fila de espera;

e) Listar as características do primeiro avião da fila.

Considere que os aviões possuem um nome e um número inteiro como identificador. Adicione outras características conforme

achar necessário.

```
#include <stdio.h>

#include <stdlib.h>

struct aviao{

   char nome[50]; //Nome do avião

   char destino [50]; //Destino do voo

   int codigo; //código cia

   char cia[50]; //nome da cia aérea

   struct item *proximo;

};

typedef struct aviao item;


item *criaFila(){

   item *fila=(item*)malloc(sizeof(item));

   fila->proximo = NULL;

   return fila;

}


int filaVazia(item *p)
```

```
{
   if(p->proximo==NULL){
      return 1;
   } else {
      return 0;
   }
}

//colocaNaFila
//Adicionar um avião à fila de espera
void queue(item *fila)
{
   item *novo=(item*)malloc(sizeof(item));
   printf("Informe o nome do aviao: ");
   scanf("%s", &novo->nome);
   printf("Informe o código: ");
   scanf("%i", &novo->codigo);
   printf("Informe a cia aérea: ");
   scanf("%s", &novo->cia);
   printf("Informe o destino: ");
   scanf("%s", &novo->destino);
   novo->proximo = NULL;

   if(filaVazia(fila)==1){
      fila->proximo=novo;
   } else {
      item *aux = fila->proximo;
      while(aux->proximo != NULL){
         aux = aux->proximo;
      }
```

```
        aux->proximo = novo;
    }
}

//Autorizar a decolagem do primeiro avião da fila
//tiraDaFila
void unqueue(item *fila)
{
  if(filaVazia(fila)==0){
        item *aux = fila->proximo;
        fila->proximo = aux->proximo;
        printf("Dados do aviao autorizado: \n Nome: %s. Código: %i. Cia: %s. Destino: %s \n", aux->nome, aux->codigo, aux->cia, aux->destino);
        free(aux);
  }
}

//Listar as características do primeiro avião da fila
void primeiro(item *fila)
{
  if(filaVazia(fila)==0){
        item *aux = fila->proximo;
        printf("Dados avião (primeiro da fila): %s. Código: %i. Cia: %s. Destino: %s \n", aux->nome, aux->codigo, aux->cia, aux->destino);
  }
}

//Listar o número de aviões aguardando na fila de decolagem
void mostrar(item *fila)
{
```

```
   item *aux = fila->proximo;
   while (aux != NULL)
   {
   printf("Avião: %s. Código: %i. Cia: %s. Destino: %s \n", aux->nome, aux->codigo, aux->cia,
aux->destino);
   aux = aux->proximo;
   }
}

void contar(item *fila)
{
   int contador = 0;
   item *aux = fila->proximo;
   while(aux != NULL){
      contador++;
      aux = aux->proximo;
   }
   printf("Numero de Avioes na Fila de espera : %i \n", contador);
}

int main(int menu, int opcao) {
   item *fila = criaFila();

   printf("**Simulação de controle de uma pista de decolagem**");
   do
   {
   printf("\n------------------------------------------------------------");
   printf("\n[1] Add um avião na fila de espera\n");
   printf("[2] Mostrar os aviões aguardando na fila de espera\n");
   printf("[3] Autorizar a decolagem do primeiro aviao\n");
```

```
   printf("[4] Numero de avioes aguardando na fila de decolagem. \n");
   printf("[5] Dados do primeiro aviao da fila.");
   printf("\n\nInsira a opção desejada: ");
   scanf("%i", &menu);
   switch(menu)
   {
   case 1: queue(fila); break;
   case 2: mostrar(fila); break;
   case 3: unqueue(fila); break;
   case 4: contar(fila); break;
   case 5: primeiro(fila); break;
   default: printf("Opcao Invalida\n"); break;
   }
   printf("Deseja continuar? [1] Sim [2] Nao:");
   scanf("%i", &opcao);
   }
   while(opcao == 1);
   return 0;
}
```

4-Implemente uma pilha dinamica que simule o historico de navegação de um browser, onde a cada pagina acessada uma nova

url e adicionada ao topo da pilha, implemente as seguintes funções:

a)adicionar nova url ao topo da pilha

b)mostrar url atual

c)voltar, removendo o topo da pilha

d)quantidade de urls adicionadas a pilha

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
struct item {
   char historico[20];
   char dia[20];
   struct item *proximo; //elemento atual vai ter o endereço do próximo item.
} typedef item;

struct pilha {
   struct item *topo; //url que está no topo da pilha
} typedef pilha;

pilha *criarPilha(){
   pilha *p = (pilha*)malloc(sizeof(pilha));
   p->topo = NULL;
   return p;
}

//adicionar nova url ao topo da pilha
void push(pilha *p){

   item *novo = (item*)malloc(sizeof(item)); //criar novo item
   printf("Informe a url: ");
   scanf("%s", &novo->historico);
   printf("Informe o dia de acesso (dd/mm/yyyy):");
   scanf("%s", &novo->dia);
   novo->proximo = NULL;

   //pilha vazia
   if(p->topo==NULL){
      p->topo = novo;
```

```
   }else{
      item *aux =  p->topo;
      novo->proximo = aux;
      p->topo = novo;
   } printf("Nova url %s, %s => %x \n", novo->historico, novo->dia, novo);
}

//voltar, removendo o topo da pilha
void pop(pilha *p){
   if(p->topo==NULL){
   printf("Pilha vazia.\n");
   }else{
      printf("URL Removida: %s, Data de Acesso: %s => %x \n", p->topo->historico, p->topo->dia, p->topo);
      item *aux = p->topo->proximo;
      free(p->topo); //remove da memoria
      p->topo = aux;
    }
}
//mostrar url atual
void mostrar(pilha *p){
   printf("Topo do histórico: %s, Data de Acesso: %s => %x \n", p->topo->historico, p->topo->dia, p->topo);
}

//quantidade de urls adicionadas a pilha
void contar(pilha *p){
   int contador = 1;
   if(p->topo==NULL){
      printf("Historico Vazio \n");
```

```
    }else{
        item*aux = p->topo;
        while(aux->proximo!=NULL){
            contador++;
            aux = aux->proximo;
        }printf("Total de URLs: %i\n", contador);
    }
}

int main(int menu, int opcao) {
    pilha *p = criarPilha();

    printf("**Simulação do Histórico de Navegacao**");
    do{
    printf("\n----------------------------------------------------------------");
    printf("\n[1] Adicionar uma url: \n");
    printf("[2] URL Atual: \n");
    printf("[3] Remover ultima url: \n");
    printf("[4] Quantidade de urls: \n");
    printf("\nInsira a opcao desejada: ");
    scanf("%i", &menu);

    switch(menu){
    case 1: push(p); break;
    case 2: mostrar(p); break;
    case 3: pop(p); break;
    case 4: contar(p); break;
    default: printf("Opcao Invalida\n"); break;
    }
    printf("\n\nDeseja continuar? [1] Sim [2] Não: ");
```

```
    scanf("%i", &opcao);
    }
    while(opcao == 1);

    return 0;
}
```

P1 – ATIVIDADE DE RECUPERAÇÃO – ESTRUTURA DE DADOS

Dada a seguinte TAD Ficha{int codigo, char nome[50], char telefone[30], int idade}

Implemente uma fila com esta TAD

```
#include <stdio.h>
#include <stdlib.h>
struct Ficha{
    int codigo;
    char nome[50];
    char telefone[30];
    int idade;
    struct Ficha *proximo;
} typedef Ficha;
struct fila{
    Ficha *primeiro;
} typedef fila;

//queue
void colocarFila(fila *f){
    Ficha *novo = (Ficha*)malloc(sizeof(Ficha));
    printf("Informe o código: ");
    scanf("%i", &novo->codigo);
    printf("Informe o nome: ");
    scanf("%s", &novo->nome);
```

```
   printf("Informe o telefone: ");
   scanf("%s", &novo->telefone);
   printf("Informe a idade: ");
   scanf("%i", &novo->idade);
   novo->proximo = NULL;
   if(f->primeiro==NULL){
      f->primeiro = novo;
      printf("Primeiro da fila %x \n", novo);
   } else {
      //procurando o fim da fila o ultimo item tem o atributo proximo = NULL
      Ficha *aux = f->primeiro;
      while(aux->proximo != NULL){
         aux = aux->proximo;
      }
      aux->proximo = novo;
      printf("Ultimo da fila %x \n", novo);
   }
}
//unqueue
void tirarFila(fila *f){
   if(f->primeiro == NULL){
      printf("Fila vazia \n");
   } else {
      printf("Item removido: Código: %i. Nome: %s . Telefone: %s. Idade: %i. => %x \n", f-
>primeiro->codigo, f->primeiro->nome, f->primeiro->telefone, f->primeiro->idade, f-
>primeiro);
      Ficha *aux = f->primeiro->proximo;
      free(f->primeiro);
      f->primeiro = aux;
   }
```

```
}

void contar(fila *f){ //contar
   int cont=1;
   if(f->primeiro == NULL){
      printf("Fila Vazia!\n");
   } else{
      Ficha *aux = f->primeiro;
      while(aux != NULL){
         cont++;
         aux = aux->proximo;
      }
      printf("Total de itens: %i \n", cont);
   }
}

void listar(fila *f){
   if(f->primeiro==NULL){
      printf("\nFila Vazia !!");
   } else {
      Ficha *aux = f->primeiro;
      do {
         printf("\nCódigo: %i. Nome: %s . Telefone: %s. Idade: %i. => %x \n", aux->codigo, aux->nome, aux->telefone, aux->idade, aux);
         aux = aux->proximo;
      } while(aux != NULL);
   }
}
int main(int argc, char *argv[]) {
   fila f;
```

```
    colocarFila(&f);
    colocarFila(&f);
    listar(&f);
    tirarFila(&f);
    listar(&f);
    contar(&f);
    return 0;
}
```

Dada a seguinte TAD Ficha{int codigo, char nome[50], char telefone[30], int idade}

Implemente uma pilha com esta TAD

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Ficha {
    int codigo;
    char nome[50];
    char telefone[30];
    int idade;
    struct Ficha *proximo;
} typedef Ficha;

struct pilha {
    struct Ficha *topo;
} typedef pilha;

int vazia(pilha *p){
    if(p->topo==NULL){
        return 1; //true
```

```
    } else{
        return 0; //false
    }
}

void push(pilha *p){
    Ficha *novo = (Ficha*)malloc(sizeof(Ficha)); //criar novo item
    printf("Informe o código: ");
    scanf("%i", &novo->codigo);
    printf("Informe o nome: ");
    scanf("%s", &novo->nome);
    printf("Informe o telefone: ");
    scanf("%s", &novo->telefone);
    printf("Informe a idade: ");
    scanf("%i", &novo->idade);

    novo->proximo = NULL;

    //pilha vazia
    if(vazia(p)){
        p->topo = novo;
    } else { //quando já temos cartas no baralho
        Ficha *aux = p->topo;
        novo->proximo = aux;
        p->topo = novo;
    }
    printf("Novo Item: %i, %s, %s, %i => %x \n", novo->codigo, novo->nome, novo->telefone,
novo->idade, novo);
}
```

```
void pop(pilha *p){
   //printf("pop \n");
   if(vazia(p)){
      printf("Pilha Vazia \n");
   } else {
      printf("Item removido: Código: %i. Nome: %s . Telefone: %s. Idade: %i. => %x \n", p->topo->codigo, p->topo->nome, p->topo->telefone, p->topo->idade, p->topo);
      Ficha *aux = p->topo->proximo;
      free(p->topo);
      p->topo = aux;
   }
}

void contar(pilha *p){
   int cont=1; //contador
   if(vazia(p)){
      printf("Pilha Vazia \n");
   } else {
      Ficha *aux = p->topo;
      while(aux->proximo!=NULL){ //enquanto o proximo for diferente de nulo
         cont++;
         aux=aux->proximo;
      }
      printf("Total de itens na pilha: %i \n", cont);
   }
}

void listar(pilha *p){
   printf("Topo da pilha = Código: %i. Nome: %s . Telefone: %s. Idade: %i. => %x \n", p->topo->codigo, p->topo->nome, p->topo->telefone, p->topo->idade, p->topo);
```

```
}

int main(int argc, char *argv[]) {
   pilha p;
   push(&p);
   push(&p);
   push(&p);
   pop(&p);
   pop(&p);

   listar(&p);
   contar(&p);

   return 0;
}
```

P2 – ESTRUTURA DE DADOS

Crie uma lista duplamente encadeada que implemente dois TADs: um para armazenar os dados de um paciente de hospital que deve conter {documento, nome, idade, gênero} e outro TAD ficha {data, medico, obs} implemente todos os métodos de uma lista. (2,0)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct paciente{
int documento;    char
nome[100];
   int idade;    char
genero[1];      struct
```

```
paciente *anterior;     struct
paciente *proximo;
} Paciente; typedef
struct ficha{     char
data[12];     char
medico[50];     char
observacao[100];
struct ficha *anterior;
struct ficha *proximo;
}     Ficha;     typedef
struct              lista{
Paciente
*pacientePrimeiro;
   Ficha *fichaPrimeiro;
}Lista ;


void colocarNoInicioPaciente(Lista *lista, int documento, char *n, int idade, char *g){
Paciente *novo=(Paciente*)malloc(sizeof(Paciente));
          novo->documento = documento;
strcpy(novo->nome, n);              novo->idade
= idade;            strcpy(novo->genero, g);
novo->anterior = NULL;              novo-
>proximo = NULL;

          if(lista->pacientePrimeiro==NULL){                    lista-
>pacientePrimeiro = novo;
          } else {
                    Paciente *antigo = lista->pacientePrimeiro;
```

```
                novo->proximo = antigo;
                antigo->anterior = novo;

                lista->pacientePrimeiro = novo;
        }
}
void colocarNoInicioFicha(Lista *lista, char *med, char *data, char *obs){
Ficha *novo=(Ficha*)malloc(sizeof(Ficha));      strcpy(novo->medico,
med);   strcpy(novo->data, data);       strcpy(novo->observacao, obs);
        novo->anterior = NULL;
        novo->proximo = NULL;           if(lista-
>fichaPrimeiro==NULL){
                lista->fichaPrimeiro = novo;
        } else {
                Paciente *antigo = lista->fichaPrimeiro;

                novo->proximo = antigo;
                antigo->anterior = novo;

                lista->fichaPrimeiro = novo;
        }
}
void colocarNoFimPaciente(Lista *lista, int documento, char *n, int idade, char *g){
Paciente *novo=(Paciente*)malloc(sizeof(Paciente));    novo->documento =
documento;      strcpy(novo->nome, n);          novo->idade = idade;
strcpy(novo->genero, g);        novo->anterior = NULL;          novo->proximo =
NULL;           if(lista->pacientePrimeiro==NULL){              lista-
>pacientePrimeiro = novo;
```

```
        } else {
                Paciente *aux = lista->pacientePrimeiro;
while(aux->proximo != NULL){
                        aux = aux->proximo;
                }
                novo->anterior = aux;           aux-
>proximo = novo; }

void colocarNoFimFicha(Lista *lista, char *med, char *data, char *obs){
Ficha *novo=(Ficha*)malloc(sizeof(Ficha));      strcpy(novo->medico,
med);   strcpy(novo->data, data);       strcpy(novo->observacao, obs);
novo->anterior = NULL;          novo->proximo = NULL;          if(lista-
>fichaPrimeiro==NULL){
                lista->fichaPrimeiro = novo;
        } else {
                Ficha *aux = lista->fichaPrimeiro;
while(aux->proximo != NULL){
                        aux = aux->proximo;
                }
                novo->anterior = aux;           aux-
>proximo = novo;
        }
}
void removerDoInicioPaciente(Lista *lista){     if(lista-
>pacientePrimeiro==NULL){
                printf("Lista vazia!\n");
        } else {
                Paciente *aux = lista->pacientePrimeiro;
                aux->proximo->anterior = NULL;                  lista-
>pacientePrimeiro = aux->proximo;
```

```
                free(aux);
        }
}
void removerDoInicioFicha(Lista *lista){
        if(lista->fichaPrimeiro==NULL){
                printf("Lista vazia!\n");
        } else {
                Ficha *aux = lista->fichaPrimeiro;
                aux->proximo->anterior = NULL;
                lista->fichaPrimeiro = aux->proximo;
                free(aux);
        }
}
void removerFimPaciente(Lista *lista){
        if(lista->pacientePrimeiro==NULL){
printf("Lista vazia!\n");
        } else {
                Paciente *aux = lista->pacientePrimeiro;
                while(aux->proximo!=NULL){
        aux = aux->proximo;
                }
                Paciente *penultimo = aux->anterior;
penultimo->proximo = NULL;              printf("Removido o
Ultimo!");
                free(aux);
        }
}
void removerFimFicha(Lista *lista){
```

```
        if(lista->fichaPrimeiro==NULL){
printf("Lista vazia!\n");
        } else {
                Ficha *aux = lista->fichaPrimeiro;
                while(aux->proximo!=NULL){
        aux = aux->proximo; }

                Ficha *penultimo = aux->anterior;
penultimo->proximo = NULL;              printf("Removido o
Ultimo!");
                free(aux);
        }
}
void removerPaciente(Lista *lista, int doc){
printf("\nRemove a Pesquis: ");          if(lista-
>pacientePrimeiro==NULL){
                printf("Lista vazia!\n");
        } else {
                Paciente *aux = lista->pacientePrimeiro;                  while(aux-
>proximo!=NULL){
                        if(doc==aux->documento){
if(aux->anterior==NULL){                                        lista-
>pacientePrimeiro= aux->proximo;                                        lista-
>pacientePrimeiro = NULL;
                                } else {
                                        aux->anterior->proximo = aux->proximo;
                                }
                                printf("%i \n", aux->documento);
                                free(aux);
                                break;
```

```
                        } else {
aux = aux->proximo;
                        }
                }

        }
} void removerFicha(Lista *lista, char
*obs){
        if(lista->fichaPrimeiro==NULL){
printf("Lista vazia!\n");
        } else {
                Ficha *aux = lista->fichaPrimeiro;              while(aux-
>proximo!=NULL){                        if(*obs == *aux->observacao){
                if(aux->anterior==NULL){
lista->fichaPrimeiro = aux->proximo;
lista->fichaPrimeiro = NULL;
                                        } else {
                                                aux->anterior->proximo = aux->proximo;
                                        }
                                        printf("%s %s \n", obs, aux->observacao);
                                        free(aux);

                                        break;
                        } else {
                                        aux = aux->proximo;
                        }
                }
        }
}
```

```
void listarPaciente(Lista *lista){
	Paciente *aux = lista->pacientePrimeiro;
	while(aux != NULL){
		printf("Paciente:\n  Documento: %i,  Nome: %s, Idade: %i, Genero: %s\n", aux->documento, aux->nome, aux->idade, aux->genero);
		aux = aux->proximo;
	}
}
void listarFicha(Lista *lista){
	Ficha *aux = lista->fichaPrimeiro;
	while(aux != NULL){
		printf("Ficha: %x, \nMedico: %s, Data: %s, Obs: %s\n", aux, aux->medico, aux->data, aux->observacao);
		aux = aux->proximo;
	}
}

int main(){
   Lista paciente1 = {NULL};     Lista ficha1 = {NULL};
colocarNoFimPaciente(&paciente1, 0001, "Mariana", 29, "F");
colocarNoFimPaciente(&paciente1, 0002, "Fernando", 32, "M");
colocarNoFimPaciente(&paciente1, 0003, "Alexa", 19, "F");
colocarNoFimPaciente(&paciente1, 0004, "José", 18, "M");
listarPaciente(&paciente1);     removerPaciente(&paciente1,
0002);     listarPaciente(&paciente1);
removerFimPaciente(&paciente1);     listarPaciente(&paciente1);
removerDoInicioPaciente(&paciente1);
listarPaciente(&paciente1);     listarFicha(&ficha1);
removerDoInicioFicha(&ficha1);     removerFimFicha(&ficha1);
```

```
   listarFicha(&ficha1);
colocarNoInicioPaciente(&paciente1,0003, "Alexa", 19, "F");
listarPaciente(&paciente1);
colocarNoInicioFicha(&ficha1,"Oftalmologista","23/11/2021","Ex
ame de vista");
   listarFicha(&ficha1);
return 0;
}
```

Crie uma arvore binaria de busca que armazene números reais, crie método para adicionar e remover itens, listar e pesquisar (2,0)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _arvore {
  int info;
  struct _arvore *esq;
struct _arvore *dir;
} Arvore;
Arvore* cria_arv_vazia () {
return NULL;
}
int adicionar (Arvore *a, int v){    if(a ==
NULL){
a=(Arvore*)malloc(sizeof(Arvore));       a-
>info = v;       a->esq = NULL;       a->dir =
NULL;       printf("\nAdicionando: %i\n", a-
>info);
   }
   else if(v < a->info){
```

```
            printf("<- ");        a->esq
= adicionar(a->esq,v); }
    else{
            printf("-> ");
       a->dir = adicionar(a->dir,v);
   }
   return a;
}

void listar (Arvore *a)
{
   if (!(a == NULL))
   {
             printf("%i", a->info);
 listar(a->esq);                listar(a-
>dir);
   }
}
int buscar(Arvore *a, int v){
   int nivel;     if
(a!=NULL){
                if(a->info==v){
                   printf("\nEncontrado: %i \n", v);
       return 0;        } else {
                    int alt_esq, alt_dir;
   if((a->info)<v)                {
alt_dir = buscar(a->dir,v);
nivel = alt_dir + 1;                return
nivel;
```

```
                }           else                    {
alt_esq = buscar(a->esq, v);
nivel = alt_esq + 1;                return
nivel;
                }
                 }
   }
   return 0;
}
int remover (Arvore *a, int v){
if(a == NULL){        return
NULL;
   }
   else{        if(a->info >v){           a-
>esq = remover (a->esq,v);
printf("\nRemovido: %i \n", v);
      }
      else if (a->info <v){           a-
>dir = remover (a->dir,v);
      }
      else{
         if((a->esq == NULL) && (a->dir == NULL)){
            free(a);
a=NULL;
         }
         else if(a->dir == NULL){
Arvore *tmp = a;
            a = a -> esq;
free (tmp);
```

```
        }
        else if(a->esq == NULL){
Arvore *tmp = a;
            a = a -> dir;
free (tmp);
        }
else{
            Arvore *tmp = a->esq;
while(tmp->dir != NULL){                tmp=tmp-
>dir;
            }
            a->info = tmp->info;
tmp->info = v;              a->esq =
remover(a->esq,v);
        }
      }
   }
   return a;
}
int main ()
{
int n, x, y;
int op, confirma;
Arvore *a = cria_arv_vazia(); do
{
   printf("Escolha a opção desejada: 1-Adicionar; 2-Listar, 3-Buscar, 4-Remover:");
scanf("%i", &op);

   switch (op)
   {
```

```
        case 1:
        printf("Digite o número:");
scanf("%i", &n);        adicionar
(a, n); break;        case 2:
        listar (a); break;
case 3:
        printf("Qual elemento deseja buscar:");
        scanf("%i", &x);
buscar(a,x); break;
        case 4:
        printf("Qual elemento deseja remover?");
        scanf("%i", &y);
remover(a,y); break;
   }
            printf("Deseja fazer uma nova transação? 1-Sim 2-Não:");
scanf("%i", &confirma);
}
            while(confirma == 1);
            return 0;
}
```