```
}

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;
}
```