**ASSOCIAÇÃO EDUCACIONAL E ASSISTENCIAL SANTA LÚCIA**

**FACULDADE SANTA LÚCIA**

**GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO**

**Enviar o trabalho da disciplina para o seguinte endereço**

comp21.estrutdados@santalucia.br

**DISCIPLINA**

**ESTRUTURA DE DADOS**

**Isabella Adorno Malvezzi**

**Mogi Mirim**

**2021**

**Isabella Adorno Malvezzi**

**DISCIPLINA ESTRUTURA DE DADOS**

Trabalho apresentado a Faculdade de Santa Lúcia, da Associação Educacional e Assistencial Santa Lúcia, como requisito parcial para a aprovação na disciplina de Estrutura de Dados do curso de Engenharia da Computação.

**Orientadora: Prof. M.Sc. Marcos Roberto de Moraes**

**Mogi Mirim**

**2021**

**SUMÁRIO**

[1. APRESENTAÇÃO DA ATIVIDADE 1](#__RefHeading___Toc34996934)

[2. APRESENTAÇÃO DO CONTEÚDO DESENVOLVIDO 4](#__RefHeading___Toc34996935)

[REFERÊNCIAS 5](#__RefHeading___Toc34996936)

1. APRESENTAÇÃO DA ATIVIDADE DO PROFESSOR

Nesta seção apresento as atividades a serem desenvolvidas.

Os exercícios práticos devem ser desenvolvidos na Linguagem C, e o código de cada exercício deverá ser entregue impresso, impreterivelmente até o dia 20 de maio, considere como informado no calendário a possibilidade do desenvolvimento no momento de aula do dia **13 de maio de 2021**.

Os exercícios teóricos devem se embasados com fontes citadas.

**Exercícios**

1. **Projete um lista de contatos para guardar informações sobre pessoas, conforme estruturas definidas abaixo:**

|  |
| --- |
| **struct contato{**  char nome[51];  char email[61];  char fone[22];  **};**  **struct lista{**  Contato \* pessoa;  Lista \* prox;  **};** |

Implemente um programa principal, que permita cadastrar e mostrar (listar os contatos).

**contato.h**

typedef struct contact Contact;

typedef struct list List;

List \* create\_contact\_list();

List \* insert\_contact(Contact \* c, List \*l);

void print\_list(List \*l);

**contato.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include "contato.h"

struct contact{

char name[51];

char email[61];

char phone[22];

};

struct list{

Contact \* person;

List \* next;

};

List \* create\_contact\_list(){

return NULL;

}

List \* insert\_contact(Contact \* c, List \* l){

List \* no = (List \*)malloc(sizeof(List));

no->person = (Contact \*)malloc(sizeof(Contact));

strcpy(no->person->name, c->name);

strcpy(no->person->phone, c->phone);

strcpy(no->person->email, c->email);

no->next = l;

return no;

}

void print\_list(List \* l){

List \* p = l;

int counter = 1;

if(p == NULL){

printf("Lista vazia :(");

return;

}

while(p!=NULL){

printf("\n------ Contato nº %d: ------\n", counter);

printf("Nome: %s", p->person->name);

printf("Email: %s", p->person->email);

printf("Telefone: %s", p->person->phone);

printf("------------------------------\n");

counter++;

p = p->next;

}

}

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include "contato.h"

struct contact{

char name[51];

char email[61];

char phone[22];

};

void new\_contact();

void pause();

List \* list;

int main(){

list = create\_contact\_list();

int op;

do {

system("clear");

printf("\n--- Cadastro de contatos ---\n");

printf("Opções: \n");

printf("1- Cadastrar novo contato\n");

printf("2- Imprimir lista de contatos\n");

printf("3- Sair\n");

printf("Escolha sua opção: ");

scanf("%d", &op);

switch(op){

case 1:

new\_contact();

break;

case 2:

system("clear");

print\_list(list);

pause();

break;

case 3:

printf("\nSaindo... \n");

break;

default:

printf("Opção invalida.\n");

pause();

break;

}

} while(op != 3);

return 0;

}

void new\_contact(){

printf("\n--- Novo contato ---\n");

Contact \* c = (Contact \*)malloc(sizeof(struct contact));

printf("Nome: ");

setbuf(stdin, NULL);

fgets(c->name, 51, stdin);

printf("Telefone: ");

setbuf(stdin, NULL);

fgets(c->phone, 22, stdin);

printf("Email: ");

setbuf(stdin, NULL);

fgets(c->email, 61, stdin);

list = insert\_contact(c, list);

printf("\nNovo contato cadastrado.");

pause();

}

void pause() {

printf("\nPressione ENTER para continuar...");

setbuf(stdin, NULL);

getchar();

}

**2. Projete uma estrutura do tipo fila (FIFO) sequencial para guardar informações de pedidos de impressão. Essas informações estarão contidas na seguinte struct:**

struct impressoraDoc{

int numreg;

char nomeimpressora[51];

};

typedef struct impressora ImpressoraDoc;

Ao final do projeto, seu programa deve rodar as seguintes funções:

1. criar a fila vazia;
2. inserir dinamicamente os seguintes itens:

|  |  |
| --- | --- |
| Numreg | Nome da Impressora |
| 1 | DiretoriaPJ |
| 2 | DiretoriaPF |
| 3 | Secretaria1 |
| 4 | Secretaria2 |

1. remover dois itens da fila;
2. apresentar como resultado os dados que estão na fila neste momento.

**impressora.h**

typedef struct fila Fila;

typedef struct lista Lista;

typedef struct impressoraDoc ImpressoraDoc;

Fila \* criar\_fila();

void inserir(Fila \* f, ImpressoraDoc \* doc);

void remover(Fila \* f);

int vazia(Fila \* f);

void liberar(Fila \* f);

void imprimir(Fila \* f);

**impressora.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include "impressora.h"

struct impressoraDoc {

int numReg;

char nomeImpressora[51];

};

struct lista {

ImpressoraDoc \* doc;

Lista \* prox;

};

struct fila {

Lista \* ini;

Lista \* fim;

};

Fila \* criar\_fila() {

Fila \* f = (Fila \*)malloc(sizeof(Fila));

f->ini = NULL;

f->fim = NULL;

return f;

}

void inserir(Fila \* f, ImpressoraDoc \* doc) {

Lista \* no = (Lista \*)malloc(sizeof(Lista));

no->doc = doc;

no->prox = NULL;

if(f->fim != NULL) {

f->fim->prox = no;

} else {

f->ini = no;

}

f->fim = no;

}

int vazia(Fila \* f) {

return (f->fim == NULL);

}

void remover(Fila \* f) {

Lista \* temp;

if(vazia(f)) {

printf("Fila vazia");

return;

}

temp = f->ini;

f->ini = temp->prox;

if(f->ini == NULL) {

f->fim = NULL;

}

free(temp);

printf("Elemento removido!\n");

}

void imprimir(Fila \* f) {

Lista \* p = f->ini;

if(vazia(f)){

printf("Fila vazia");

return;

}

printf("NumReg \t Nome da Impressora\n");

while(p != NULL) {

printf("%d \t %s", p->doc->numReg, p->doc->nomeImpressora);

p = p->prox;

}

}

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "impressora.h"

struct impressoraDoc {

int numReg;

char nomeImpressora[51];

};

void nova\_impressao(int \* reg);

void pause();

Fila \* fila;

int main(){

fila = criar\_fila();

// Achei interessante manter o numReg como um contador interno, pro usuário não precisar ficar digitando

int reg = 1;

int op;

do {

system("clear");

printf("\n--- Fila de impressão de documentos ---\n");

printf("Opcoes: \n");

printf("1- Adicionar nova impressão à fila\n");

printf("2- Mostrar fila\n");

printf("3- Remover item da fila\n");

printf("4- Sair\n");

printf("Escolha sua opção: ");

scanf("%d", &op);

switch(op){

case 1:

nova\_impressao(&reg);

break;

case 2:

system("clear");

imprimir(fila);

pause();

break;

case 3:

remover(fila);

pause();

case 4:

printf("\nSaindo... \n");

break;

default:

printf("Opcao invalida\n");

pause();

break;

}

} while(op != 4);

return 0;

}

void nova\_impressao(int \* reg) {

printf("\n--- Nova impressão ---\n");

ImpressoraDoc \* doc = (ImpressoraDoc \*)malloc(sizeof(struct impressoraDoc));

printf("Nome da impressora: ");

setbuf(stdin, NULL);

fgets(doc->nomeImpressora, 51, stdin);

doc->numReg = \*reg;

\*reg = \*reg + 1;

inserir(fila, doc);

printf("\nNova impressão adicionada a fila.");

pause();

}

void pause() {

printf("\nPressione ENTER para continuar...");

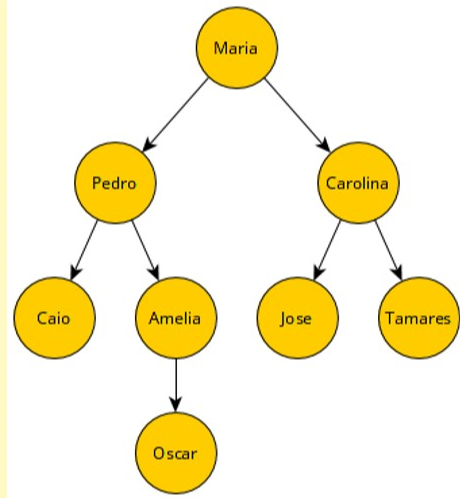
setbuf(stdin, NULL);

getchar();

}

**3. Em linguagem C, crie uma árvore binária que represente a família do Sr. Oscar (Bisavô de Maria). Você deve criar os arquivos com as funções responsáveis execução (exearv.c) o protótipo da função (exearv.h) e o arquivo para rodas as instruções abaixo.**

1. Criar uma arvore vazia.
2. Inserir os seguintes em cada nó:



1. Considerando a árvore anterior, faça uma função que retorne se há na árvore da família alguém com o nome "Tamares" e o grau de parentesco que há com a "Maria" bisneta do Sr. Oscar. Parentesco: 1 -geração atual, 2 -pais, 3 -avós, 4 -bisavós.

**exearv.h**

typedef struct family Family;

Family \* create\_empty();

Family \* create(char \* name, Family \* left, Family \* right);

int empty(Family \* f);

void print\_member(char \* name, int spaces);

void print\_family(Family \* f, int spaces);

Family \* free\_family(Family \* f);

void find(Family \* f, char \* member, int kin);

void get\_kin(int kin);

**exearv.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "exearv.h"

struct family {

char name[51];

Family \* left;

Family \* right;

};

Family \* create\_empty() {

return NULL;

}

Family \* create(char \* name, Family \* left, Family \* right) {

Family \* f = (Family \*)malloc(sizeof(Family));

strcpy(f->name, name);

f->left = left;

f->right = right;

return f;

}

int empty(Family \* f) {

return (f == NULL);

}

void print\_member(char \* name, int spaces) {

for (int i = 0; i < spaces; i++) {

printf(" ");

}

printf(" %s \n", name);

}

void print\_family(Family \* f, int spaces) {

if(empty(f)) {

print\_member(" -\*- ", spaces);

return;

}

print\_family(f->right, spaces + 6);

print\_member(f->name, spaces);

print\_family(f->left, spaces + 6);

}

Family \* free\_family(Family \* f) {

if(!empty(f)) {

free\_family(f->right);

free\_family(f->left);

free(f);

}

return NULL;

}

void find(Family \* f, char \* member, int kin) {

if(empty(f)) {

return;

}

kin++;

if(strcmp(f->name, member) == 0) {

printf("\nGrau de parentesco de %s: %d ", member, kin);

get\_kin(kin);

return;

}

find(f->left, member, kin);

find(f->right, member, kin);

}

void get\_kin(int kin) {

switch (kin) {

case 1:

printf("(geração atual)\n");

break;

case 2:

printf("(pais)\n");

break;

case 3:

printf("(avós)\n");

break;

case 4:

printf("(bisavós)\n");

break;

default:

break;

}

}

**main.c**

#include <stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include "exearv.h"

int main(){

Family \* bisavo = create("Oscar", create\_empty(), create\_empty());

Family \* vo1 = create("Amelia", create\_empty(), bisavo);

Family \* vo2 = create("Caio", create\_empty(), create\_empty());

Family \* vo3 = create("Tamares", create\_empty(), create\_empty());

Family \* vo4 = create("Jose", create\_empty(), create\_empty());

Family \* pai = create("Pedro", vo2, vo1);

Family \* mae = create("Carolina", vo4, vo3);

Family \* maria = create("Maria", pai, mae);

print\_family(maria, 10);

int kin = 0;

char member[51] = "Tamares";

find(maria, member, kin);

}

**Nota 1: Os itens de 01 a 03 devem ser apresentados como códigos.**

**4. Elabore um texto comparativo sobre os seguintes métodos de busca: sequencial não ordenada, sequencial ordenada e binária (considere o algoritmo já ordenado para essa última). Neste comparativo apresente razões justificadas para uso de uma ou de outra estrutura para ordenação de vetores de diferentes tamanhos. Analise os casos de pior caso e melhor caso de cada um deles.**

Na busca linear, o elemento buscado é comparado com todos aqueles que existem no array. Caso ele esteja na terceira posição, por exemplo, a busca não terá um custo tão grande, já que a estrutura pode ter um "return" dentro de si que encerra a busca quando o valor é encontrado. Porém, caso o elemento desejado esteja na última posição (ou nem exista), e o vetor possuir 500000 elementos, esta será muito custosa, gerando o “Big Oh” (o número de iterações será igual ao número de valores do vetor).

Já na busca linear com um vetor ordenado, é possível realizar um controle maior de quando a busca será encerrada: caso o usuário entre com o valor 10, e já na primeira posição do array exista o número 30, esta será finalizada, já que 10 é menor do que 30, e, neste vetor, o menor número era 30, então não existe o número 10 dentro dele. Porém, caso o número procurado seja muito grande, ainda existe a chance dele estar na última posição do array, gerando novamente o Big Oh.

Para arrays com muitos valores, o ideal é a utilização da busca binária (ordenada): nela, o vetor é dividido pela metade diversas, comparando sempre o item procurado com o valor do meio daquela seção e, caso não sejam iguais, a busca é continuada com a metade inferior ou posterior do vetor, dependendo se o número do meio é maior ou menor do que o procurado. O custo computacional nunca será o O(n), uma vez que grande parte do array nem será analisado durante a busca, tornando-a muito mais eficiente que a busca linear ordenada nesse sentido.

Qualquer dúvida pode ser reportada pelo e-mail: [professormoraes@gmail.com](mailto:professormoraes@gmail.com). A entrega deve ser feita no e-mail constante na capa deste trabalho **(comp21.estrutdados@santalucia.br)**, até no máximo em **20 de maio de 2021.**

**REGRA GERAL DE FORMATAÇÃO PARA ELABORAÇÃO DA ATIVIDADE- Seguir o manual de metodologia da faculdade disponível na área do discente**

- Folha A4 (21,0 cm X 29,7 cm), em papel branco e tinta de cor preta;

**Margem**

a) Margem esquerda: 3,0 cm

b) Margem direita: 2,0 cm

c) Margem superior: 3,0 cm

b) Margem inferior: 2,0 cm

**Espaçamento entre linhas e parágrafos**

- Texto normal – espaço 1,5 linha;

**Tipo e tamanho da fonte**

- Texto normal: Arial, 12;

2. APRESENTAÇÃO DO CONTEÚDO DESENVOLVIDO PELO ALUNO

Inicie aqui a solução da atividade solicitada.

# REFERÊNCIAS[[1]](#footnote-2)

Todas as referências citadas no texto, devem ser apresentadas aqui em ordem alfabética por sobrenome de autor.

1. Baseadas na norma NBR 6023, de 2018, da Associação Brasileira de Normas Técnicas (ABNT). [↑](#footnote-ref-2)