**INTRODUÇÃO**

Os problemas enfrentados no ano de 2020 causados pelo coronavírus, não só no Brasil, mas em todo o mundo, mostrou que a gestão do fluxo de pacientes tornou-se um desafio para instituições hospitalares. A necessidade de melhorar os serviços de saúde, agilizando a adaptação da relação entre capacidade e demanda e permitindo que haja um fluxo adequando evitando gargalos, tem como necessário a busca por uma solução que ajude agilizando os serviços, para que o paciente tenha o cuidado certo na hora certa.

Para melhorar o gerenciamento nas redes de saúde, a criação de um programa que entregue a informação necessária e agilize o controle de internações, é a melhor possibilidade para que as instituições tenham dinamismo no atendimento, por isso pensar na forma de mapear os processos é necessário a criação de um programa para gestão de leitos.

Foram utilizadas bibliotecas próprias compostas da mesclagem de funções nativas da linguagem C, presentes nas bibliotecas “stdlib.h”, “stdio.h”, “locale.h”, “string.h” e “ctype.h”.

**1 - DESCRIÇÃO DO “mainFunctions.c”**

**1.1 - ESTRUTURAS**

A estrutura *“controls”* contem 3 dados do tipo inteiro, sendo eles:

**int id\_leitos:** Controle de ids gerados para leitos.

**int id\_pacientes:** Controle de ids gerados para pacientes

**int id\_equips:** Controle de ids gerados para equipamentos.

A estrutura *“matrizChar”* contem 4 tipos de dado, sendo eles:

**char \*\*\*dados:** Ponteiro de grau 3, para alocação dinâmica de matriz com dados do tipo String.

**int sizeChar:** Inteiro contendo o tamanho do vetor de caracteres de cada *“célula”* da matriz.

**int qtdCol:** Inteiro contendo a quantidade de colunas da matriz.

**int qtdLin:** inteiro contendo a quantidade de linhas da matriz.

A estrutura *“l\_leito”* contem 7 tipos de dado, sendo eles:

**int id:** Inteiro contendo o Id de cadastro do leito.

**int tipo:** Inteiro contendo o código do tipo do leito

**char nome[100]:** Vetor de caracteres contendo o nome do paciente.

**int qtdLeitos:** Inteiro contendo a quantidade de unidades desse leito.

**int qtdEquipVinculados:** Inteiro contendo a quantidade de equipamentos vinculados a este leito.

**int \*id\_equip\_vinculado:** Ponteiro para alocação dinâmica de *“array”* contendo os ids do tipo inteiro de cada equipamento vinculado.

**struct l\_leito \*next:** Ponteiro com o endereço do próximo *“nó”* dessa lista encadeada.

A estrutura *“l\_paciente”* contem 9 tipos de dado, sendo eles:

**int id:** Inteiro contendo o id de cadastro do paciente.

**int id\_leito\_vinculado:** Inteiro contendo o ido do leito em que o paciente esta vinculado.

**char cpf[15];** Vetor de caracteres contendo o número do CPF do paciente formatado.

**char nome[100]:** Vetor de caracteres contendo o nome do paciente.

**char data\_nasc[11]:** Vetor de caracteres contendo a data de nascimento.

**char data\_entrada[11]:** Vetor de caracteres contendo a data de entrada no leito.

**char data\_saida[11]:** Vetor de caracteres contendo a data de saída do leito.

**int cod\_motivo\_saida:** Inteiro contendo o código do motivo de saída do leito.

**struct l\_paciente \*next:** Ponteiro com o endereço do próximo *“nó”* dessa lista encadeada.

A estrutura *“l\_equipamentos”* contem 3 tipos de dado, sendo eles:

**int id:** Inteiro contendo o id de cadastro do equipamento.

**char nome[100]:** Vetor de caracteres contendo o nome do equipamento.

**struct l\_equipamentos \*next:** Ponteiro com o endereço do próximo *“nó”* dessa lista encadeada.

As estruturas do tipo *“leito”, “paciente” e “equipamentos”* contem os mesmos dados das estruturas com o prefixo *“l\_”* com exceção dos ponteiros. Essas estruturas são utilizadas para armazenamento dos dados em arquivos e auxiliam na coleta de dados do usuário.

**1.2 - FUNÇÕES**

**1.2.1 - ESTRUTURA DE EQUIPAMENTOS**

**1.2.1.1 - int VinculaEquipamento(l\_leito \*ref, int id\_leito, int id\_equipamento);**

Função que recebe uma lista encadeada do tipo *“l\_leito”* para realizar a alocação dinâmica de um *“array”* contendo um ID da lista encadeada do tipo *“l\_equipamentos”.*

**1.2.1.2 - l\_equipamentos \* busca\_por\_id\_equipamento(l\_equipamentos \*ref, int id);**

Função que retorna um ponteiro contendo o *“nó”* do id repassado como referencia na função. A busca é realizada no vetor de referência do tipo *“l\_equipamentos”*.

**1.2.1.3 - l\_equipamentos \* free\_equipamentos(l\_equipamentos \*ref, l\_leito \*vinculos, int id);**

Recebe como parâmetro uma lista encadeada do tipo *“l\_equipamentos”,* uma lista encadeada do tipo *“l\_leitos”* e o id do equipamento que deverá ser excluido. A função além de apagar o nó da lista, também apaga o id vinculado em todos os leitos cadastrados. É retornado a própria lista.

**1.2.1.4 - l\_equipamentos \* freeAll\_equipamentos(l\_equipamentos \*ref);**

Função de limpeza da estrutura *“l\_equipamentos”,* com o propósito de evitar vazamento de memória. Só é preciso passar a lista de referência, para que os dados sejam apagados. É retornado a própria lista.

**1.2.1.5 - l\_equipamentos \* inserir\_equipamento(l\_equipamentos \*ref, equipamentos dados);**

Insere um equipamento em uma lista encadeada do tipo *“l\_equipamentos”,* deve ser passado a lista de referencia e uma estrutura auxiliar do tipo *“equipamentos”.* É retornado a própria lista.

**1.2.1.6 - l\_equipamentos \* new\_l\_equipamentos();**

Retorna um ponteiro NULL do tipo *“l\_equipamentos”.*

**1.2.1.7 - void cleanEquipamento(equipamentos \* ref);**

Simplesmente limpa uma variável do tipo *“equipamentos”*.

**1.2.1.8 - void desvinc\_equip(l\_leito \*ref, int id\_leito, int id\_equipamento);**

Desvincula um equipamento de um leito, deve ser repassado uma lista encadeada do tipo *“l\_leito”,* o id do leito em que o equipamento será desvinculado e o id do equipamento que será desvinculado.

**1.2.2 - ESTRUTURA DE LEITOS**

**1.2.2.1 - char \* tipoLeito(int cod);**

Deve ser repassado um código do tipo inteiro, para que seja retornado a *“string”* que contém o tipo do leito.

**1.2.2.2 - l\_leito \* busca\_por\_id\_leito(l\_leito \*ref, int id);**

Função que retorna um ponteiro contendo o *“nó”* do id repassado como referência na função. A busca é realizada no vetor de referência do tipo *“l\_leito”*.

**1.2.2.3 - l\_leito \* free\_leito(l\_leito \*ref, int id);**

Recebe como parâmetro uma lista encadeada do tipo *“l\_leito”,* e o id do leito que deverá ser excluído. É retornado a própria lista.

**1.2.2.4 - l\_leito \* freeAll\_Leitos(l\_leito \*ref);**

Função de limpeza da estrutura *“l\_leito”,* com o propósito de evitar vazamento de memória. Só é preciso passar a lista de referência, para que os dados sejam apagados. É retornado a própria lista.

**1.2.2.5 - l\_leito \* inserir\_leito(l\_leito \*ref, leito node);**

Insere um leito em uma lista encadeada do tipo *“l\_leito”,* deve ser passado a lista de referência e uma estrutura auxiliar do tipo *“leito”.* É retornado a própria lista.

**1.2.2.6 - l\_leito \* new\_l\_leito();**

Retorna um ponteiro NULL do tipo *“l\_leito”.*

**1.2.2.7 - void cleanLeitos(leito \* ref);**

Simplesmente limpa uma variável do tipo *“leito”*.

**1.2.3 - ESTRUTURA DE PACIENTES**

**1.2.3.1 - char \*MotivoSaida\_Paciente(int codigo);**

Deve ser repassado um código do tipo inteiro, para que seja retornado a *“string”* que contém o motivo de saída do paciente.

**1.2.3.2 - l\_paciente \* busca\_por\_id\_paciente(l\_paciente \*ref, int id);**

Função que retorna um ponteiro contendo o *“nó”* do id repassado como referência na função. A busca é realizada no vetor de referência do tipo *“l\_paciente”*.

**1.2.3.3 - l\_paciente \* free\_paciente(l\_paciente \*ref, int id);**

Recebe como parâmetro uma lista encadeada do tipo *“l\_paciente”* e o id do leito que deverá ser excluído. É retornado a própria lista.

**1.2.3.4 - l\_paciente \* freeAll\_paciente(l\_paciente \*ref);**

Função de limpeza da estrutura *“l\_paciente”,* com o propósito de evitar vazamento de memória. Só é preciso passar a lista de referência, para que os dados sejam apagados. É retornado a própria lista.

**1.2.3.5 - l\_paciente \* inserir\_paciente(l\_paciente \*ref, paciente dados);**

Insere um paciente em uma lista encadeada do tipo *“l\_paciente”,* deve ser passado a lista de referência e uma estrutura auxiliar do tipo *“paciente”.* É retornado a própria lista.

**1.2.3.6 - l\_paciente \* new\_l\_paciente();**

Retorna um ponteiro NULL do tipo *“l\_paciente”.*

**1.2.3.7 - void cleanPaciente(paciente \* ref);**

Simplesmente limpa uma variável do tipo *“paciente”*.

**1.2.4 - ESTRUTURA MATRIZ DE STRING**

**1.2.4.1 - matrizChar \* FilterByName\_equipamentos(l\_equipamentos \*ref, char value[], int tamStr);**

Retorna um ponteiro do tipo *“matrizChar”* contendo o resultado do filtro por nome da estrutura *“l\_equipamentos”.* Deve ser repassado a lista encadeada de referência, a palavra de pesquisa e o tamanho do vetor de caracteres contido na estrutura de pesquisa.

**1.2.4.2 - matrizChar \* FilterByName\_leitos(l\_leito \*ref, char value[], int tamStr);**

Retorna um ponteiro do tipo *“matrizChar”* contendo o resultado do filtro por nome da estrutura *“l\_leito”.* Deve ser repassado a lista encadeada de referência, a palavra de pesquisa e o tamanho do vetor de caracteres contido na estrutura de pesquisa.

**1.2.4.3 - matrizChar \* FilterByName\_pacientes(l\_paciente \*ref, l\_leito \*lista, char value[], int tamStr);**

Retorna um ponteiro do tipo *“matrizChar”* contendo o resultado do filtro por nome da estrutura *“l\_paciente”.* Deve ser repassado a lista encadeada de referência, a lista encadeada do tipo *“l\_leito”*, a palavra de pesquisa e o tamanho do vetor de caracteres contido na estrutura de pesquisa.

**1.2.4.4 - void matChar\_sort(matrizChar \*ref, int criteriaCol);**

Organiza um dado do tipo *“matrizChar”* com o um código inspirado no algoritmo *“bubble sort”*. Deve ser repassado a matriz de referência junto da coluna que deve ser utilizada como critério. O retorno é uma lista organizada no formato menor para maior.

**1.2.4.5 - void print\_matrizChar(matrizChar \*ref);**

É imprimido no terminal um dado do tipo *“matrizChar”* em formato de tabela, deve ser repassado somente a matriz de referência.

**1.2.4.6 - void deleteMatrizChar(matrizChar \*ref);**

Função de limpeza do dado *“matrizChar”,* com o propósito de evitar vazamento de memória. Só é preciso passar a matriz de referência, para que os dados sejam apagados.

**1.2.5 – ARQUIVOS**

**1.2.5.1 - void fatualiza\_controls(controls \*ref, char \*nomearq);**

Recebe uma estrutura do tipo *“controls”,* e regrava as informações contidas nela em um arquivo binário. Deve ser repassado a estrutura de referência e o nome do arquivo.

**1.2.5.2 - void fcarrega\_dados\_controls(controls \*ref, char \*nomearq);**

Carrega os dados de um arquivo binário em uma estrutura única do tipo *“controls”.* Deve ser repassado a estrutura de referencia e o nome do arquivo.

**1.2.5.3 - l\_equipamentos \* fcarrega\_dados\_equip(char \*nomearq);**

Função que retorna um ponteiro contendo uma lista encadeada do tipo *“l\_equipamentos”.* Deve ser passado o nome do arquivo que contém os dados, caso contrário será retornado NULL.

**1.2.5.5 - void fatualiza\_equips(char \* nomearq, l\_equipamentos \*ref);**

Recebe uma lista encadeada do tipo *“l\_equipamentos”,* e regrava as informações contidas nela em um arquivo binário. Deve ser repassado também o nome do arquivo.

**1.2.5.5 - void fatualiza\_vincEquips(char \* nomearq, l\_leito \*ref);**

Atualiza o arquivo que contem os dados de todos os equipamentos vinculados a cada leito, os dados são guardados da seguinte forma:

{ Int id\_leito, Int qtdEquipVinculado, sizeof(int) \* qtdEquipVinculado }

É necessário repassar o nome do arquivo binário de referencia e a lista encadeada do tipo *“l\_leito”.*

**1.2.5.6 - void fcarrega\_vincEquips(char \* nomearq, l\_leito \*ref);**

Carrega os *“arrays dinâmicos”* em cada *“nó”* da lista encadeada do tipo *“l\_leito”* que contenha os ids correspondentes. É necessário repassar o nome do arquivo binário de referencia e a lista encadeada do tipo *“l\_leito”.*

**1.2.5.7 - void fatualiza\_leitos(char \* nomearq, l\_leito \*ref);**

Recebe uma lista encadeada do tipo *“l\_leito”,* e regrava as informações contidas nela em um arquivo binário. Deve ser repassado também o nome do arquivo.

**1.2.5.8 - l\_leito \* fcarrega\_dados\_leitos(char \*nomearq);**

Função que retorna um ponteiro contendo uma lista encadeada do tipo *“l\_leito”.* Deve ser passado o nome do arquivo que contém os dados, caso contrário será retornado NULL.

**1.2.5.9 - void fatualiza\_pacientes(char \* nomearq, l\_paciente \*ref);**

Recebe uma lista encadeada do tipo *“l\_paciente”,* e regrava as informações contidas nela em um arquivo binário. Deve ser repassado também o nome do arquivo.

**1.2.5.10 - l\_paciente \* fcarrega\_dados\_pacientes(char \*nomearq);**

Função que retorna um ponteiro contendo uma lista encadeada do tipo *“l\_paciente”.* Deve ser passado o nome do arquivo que contém os dados, caso contrário será retornado NULL.

**2 - DESCRIÇÃO DO “utilityFunctions.c”**

**2.1 – ESTRUTURAS**

A estrutura *“data”* contem 3 inteiros, sendo eles:

**int dia:** Inteiro contendo o dia,

**int mês:** Inteiro contendo o mês.

**int ano:** Inteiro contendo o ano.

**2.2 – FUNÇÕES**

**2.2.1 - TRATAMENTO DE DATAS**

**2.2.1.1 - char \* dataAtual();**

É retornado um ponteiro do tipo *“CHAR”* contendo a data atual no formato *“string”* com tamanho de 11 caracteres.

**2.2.1.2 - int dataValida(Data \*analise);**

Verifica se uma estrutura do tipo data contem uma data valida e retorna 1 para verdadeiro e 0 para falso.

**2.2.1.3 - Data \* str\_to\_data(char \*str, const char \*key);**

Retorna um ponteiro de uma estrutura no formato *“Data”,* deve ser repassado um vetor de caracteres contendo 11 digitos e na variável *“key”* deve ser informado qual o caractere que separa a data.

**2.2.1.4 - unsigned long dist\_dias (Data \*inicio, Data \*fim);**

Verifica a distancia em dias entre duas datas do formato *“Data”.* É retornado os dias no formato *“unsigned long”.*

**2.2.2.1.5 - int bissexto (int ano);**

Verifica se o ano passado no formato inteiro é bissexto. É retornado 1, caso seja bissexto e 0 caso não seja.

**2.2.2 - TRATAMENTOS EM GERAL**

**2.2.2.1 - char \* int\_to\_str(int value);**

Deve ser repassado um valor no formato *“int”,* para que seja retornado sua correspondente no formato de ponteiro *“char”.*

**2.2.2.2 - char \* maiuscula(char str[]);**

Recebe um vetor de caracteres e retorna sua correspondente no formato maiúsculo. É retornado um ponteiro que não é liberado após o uso!

**2.2.2.3 - char \*\* splitStr(char \*str, const char \*key);**

É retornado um ponteiro para um vetor de caracteres contendo uma *“string”* dividida em um *“array”* por meio de um caractere especial. Deve ser passado o vetor de caracteres que deseja ser separado e o critério de separação.

**2.2.2.4 - int charUsed(char value[]);**

Deve ser repassado um vetor de caracteres para que seja contado quantos caracteres estão sendo usados na *“string”.* O retorno é do tipo inteiro.

**2.2.2.5 - int intlen(int value);**

Deve ser repassado um inteiro, para que seja retornado quantos dígitos esse inteiro tem. O retorno é do tipo inteiro.

**2.2.2.6 - int userIptCtrl(char \*str, char \*keys);**

Deve ser repassado um vetor de caracteres e quais as únicas chaves que devem estar contidas. Por exemplo, se você quiser que seja digitado apenas números, no campo Keys, deve ser digitado “1234567890” e repassado no str qual a string para verificar. É retornado 1 caso seja digitado apenas os dígitos solicitados e 0 caso tenha algum caractere diferente.

**2.2.2.7 - void ClearBuffer();**

Limpa o *“Buffer”* para solicitação de nova entrada no teclado pelo usuário.

**2.2.2.8 - void swapStr(char\* a, char\* b, int sizeStrings);**

Troca a posição duas *“strings”*. Devem ser repassado as *“strings”* que deseja trocar, e o tamanho delas.

**2.2.2.9 - void strUserInput(char \*str, int len);**

Recebe o input do teclado de um usuário, funciona em conjunto com o *“fgets”*(Nativo), *“removeBrkLn”* e o *“ClearBuffer”*. Deve ser repassado o vetor de caracteres de destino juntamente do tamanho do vetor.

**2.2.2.10 - void removeBrkLn(char str[]);**

Deve ser repassado um vetor de caracteres para que seja retirado as quebras de linhas do mesmo.

**2.2.2.11 - void printEspacos(int qtd);**

Utiliza a função nativa *“printf”,* para ser impresso caracteres de espaço. Deve ser repassado a quantidade de espaços que serão impressos.

**2.2.2.12 - void pauseEXE(char message[]);**

Pausa a execução do terminal, compatível com Windows. Deve ser passado uma mensagem para o usuário.

**2.2.2.13 - void clearWindowsTerminal();**

Limpa o terminal, compatível com Windows.

**2.2.3 - TRATAMENTO DE ARQUIVOS**

**2.2.3.1 - int fCriaBD(char \*nomeArq);**

Função que cria um arquivo binário com modo de abertura *“ab”,* é retornado 1 caso tenha sido criado com sucesso e 0 caso não tenha sido criado*.* Deve ser passado o nome do arquivo.

**2.2.3.2 - int fexiste(char \* nomearq);**

Verifica se um arquivo existe e retornam 1 para verdadeiro e 0 para falso. Deve ser passado o nome do arquivo.

**2.2.3.3 - int fsavedata(void \*Dados, size\_t lenDados,char \*nomeArq);**

Salva um dado de qualquer tipo em um arquivo binário já existente. Deve ser passado o dado que devera ser salvo, o tamanho desse dado e o nome do arquivo de destino. Retorna 1 caso tenha sido salvo corretamente e 0 caso não tenha sido salvo.

**2.2.3.4 - long flen(FILE \*arquivo);**

Retorna o tamanho de um arquivo binário, deve ser passado o nome do arquivo para ser retornado no formato *“long”*, a quantidade de *“bytes”* que o arquivo possui.