Trabalho 1: hospital

Nome: Gabriel Zagury de Magalhães
Matrícula: 2210912

Objetivo: Criar um sistema de atendimentos, com níveis de prioridade, para um hospital.
funcionalidades:

- criar uma nova lista de atendimentos
- inserir um novo pedido de forma ordenada por prioridade (Vermelha > Amarela > Verde) na lista, gerando um número novo sequencial para cada pedido
- Visualizar a lista no estado atual
- Remover um pedido da lista a partir de seu número sequencial (a qualquer momento)

Estrutura:

```
int numero_associado(char *prioridade);
```
retorna o número associado a cada prioridade (ex: “Vermelha” = 1, pois é mais importante, verde = 3)

```
Lista *lst_insere(Lista *lst, char *prioridade);
insere um novo pedido de forma ordenada por prioridade (Vermelha > Amarela > Verde) na lista, a partir de uma lista (vazia ou preenchida) e uma prioridade, retorna a lista alterada
```

```
void lst_imprime(Lista *lst);
```
Imprime toda a lista, que foi recebida.

```
Lista *lst_remove(Lista *lst, int val);
```
Remove um valor de uma lista, esse valor é gerado automaticamente pela função lst_insere

Solução:

Bibliotecas usadas:

```
stdio.h
stdlib.h
string.h
```

1. criamos uma variável global para guardar o numero sequencial identificador de cada paciente (único)

```
int num = 0;
```

2. A lista encadeada que será usada

```
struct lista
{
    int numero;
    char *prioridade;
    struct lista *prox;
};
```
typedef struct lista Lista;

3. Função para criar uma lista vazia

apenas retorna um ponteiro do tipo Lista com valor NULL

```
Lista *lst_cria(void)
{
    return NULL;
}
```

4. *Função para facilitar a ordenação, retorna o número associado a uma prioridade. Caso necessário adicionar outra prioridade, adicionar aqui (e seu número associado)*

```
int numero_associado(char *prioridade)
{
    if (strcmp(prioridade, "Vermelha") == 0)
        return 1;
    if (strcmp(prioridade, "Amarela") == 0)
        return 2;
    if (strcmp(prioridade, "Verde") == 0)
        return 3;
    return 0;
}
```

5. Função de inserir um novo elemento na lista de forma ordenada

se a lista está vazia ou o novo elemento for menor que o primeiro, retorna o novo valor, apontando para os outros (novo é o primeiro valor)

se o novo não deve ser o primeiro valor, retornar lst

```
// ordem de prioridade: Vermelha, Amarela, Verde
Lista *lst_insere(Lista *lst, char *prioridade)
{
    // cria um novo nó
    Lista *novo = (Lista *)malloc(sizeof(Lista));
    int numero_da_prioridade =
numero_associado(prioridade);
    // atribui o valor ao nó
    novo->numero = num;
    num = num +
1;                                                  //
incrementa o número do próximo nó (chamada futura)
    novo->prioridade = (char *)malloc(strlen(prioridade)
+ 1); // +1 para o \0
    strcpy(novo->prioridade, prioridade);
    // verifica se a lista está vazia
    if (lst == NULL)
    {
        // se sim, o novo nó é o primeiro e único
elemento da lista
        novo->prox = NULL;
        return novo;
    }
    // se não, procura a posição correta para o novo nó
    Lista *ant = NULL; // ponteiro para o elemento
anterior
    Lista *p = lst;    // ponteiro para o elemento atual

    // percorre a lista até encontrar um elemento com
prioridade maior ou até o final da lista
    while (p != NULL && numero_associado(p->prioridade)
<= numero_da_prioridade)
    {
        ant = p; // atualiza os ponteiros
        p = p->prox;
    }
    // se a lista está vazia ou o novo elemento for
menor que o primeiro, insere no início
    if (ant == NULL)
    {
        novo->prox = lst;
        return novo;
    }
    // se não, insere no meio ou no fim
```

```
    ant->prox = novo;
    novo->prox = p;

    return lst;
}
```

6. Função para imprimir a lista

o ponteiro caminha pela lista até ela chegar no ultimo nó (null), imprimindo o seu número e prioridade

```
void lst_imprime(Lista *lst)
{
    Lista *p = lst; // ponteiro para o elemento atual
    printf("\nImprimindo: \n");
    do
    {
        printf("%d - %s\n", p->numero, p->prioridade); // imprime o elemento atual
        p = p->prox;                                   // atualiza o ponteiro para o próximo elemento
    } while (p != NULL);                               // repete até o final da lista
    return;
}
```

7. Função para remover elemento, a partir de seu número associado. Assim, podendo remover qualquer elemento em qualquer posição, há qualquer momento.

Se o elemento é o primeiro, o anterior é usado como buffer, recebendo o valor do segundo elemento, e o primeiro é liberado (free)
Se não for, o próximo do anterior aponta para o próximo do que vai ser removido. Depois, o elemento é liberado (free), assim como sua string associada (pois foi alocada dinamicamente).

```
// remove um elemento da lista
Lista *lst_remove(Lista *lst, int val)
{
    Lista *p = lst;    // ponteiro para o elemento atual
    Lista *ant = NULL; // ponteiro para o elemento anterior
    do
    {
        if (p->numero == val) // vamos remover aqui
        {
```

```
            // caso 1: elem é o primeiro
            if (ant == NULL)
            {
                ant = lst->prox; // usa ant como buffer
                free(p->prioridade);
                free(lst);
                return ant;
            }
            // caso 2: elem é o último ou está no meio
            ant->prox = p->prox;
            free(p->prioridade);
            free(p);
            return lst;
        }
        // caminha na lista
        ant = p;
        p = p->prox;
    } while (p != NULL);
    return lst;
}
```

8. Testes

```
int main(void)
{
    Lista *lst = lst_cria();
    lst = lst_insere(lst, "Amarela");
    lst = lst_insere(lst, "Vermelha");
    lst = lst_insere(lst, "Verde");
    lst = lst_insere(lst, "Amarela");
    lst = lst_insere(lst, "Vermelha");
    lst = lst_insere(lst, "Amarela");
    lst = lst_insere(lst, "Verde");
    lst_imprime(lst);
    lst = lst_remove(lst, 3);
    lst_imprime(lst);
    lst = lst_remove(lst, 1);
    lst_imprime(lst);

    lst = lst_insere(lst, "Vermelha");
    lst = lst_insere(lst, "Amarela");
    lst = lst_remove(lst, 6);
    lst_imprime(lst);
    lst = lst_remove(lst, 4);
    lst_imprime(lst);
    lst = lst_remove(lst, 2);
```

```
    lst_imprime(lst);
    return 0;
}
```

Observações e conclusões:

O programa aceita entradas incorretas na prioridade (emite um aviso e não altera a lista) e na função de remover (não remove nada). A maior dificuldade encontrada foi fazer a ordenação por prioridade, de forma que o código possa ser expandido com facilidade, aceitando novos valores de prioridade. A solução encontrada foi criar a função que retorna um número a partir da string adicionada, podendo alterar apenas o numero associado a cada string na função.
Finalmente, o programa funciona corretamente (o elemento mais recente é inserido no final da lista), o mais recente, a cabeça.