Relatório T1 INF-1010:

Diego Miranda: 2210996
Eric Goulart: 2110878

Objetivo:
O trabalho traz como objetivo a criação de um sistema que, por meio dos conhecimentos sobre Listas encadeadas adquiridos nas aulas, seja capaz de exemplificar um sistema de listagem de pacientes de um hospital, organizando-os por ordem de chegada e urgência de atendimento. Além disso, devem ser também realizadas adições e remoções de pacientes desta lista, sempre estando ordenada de acordo com grau de urgência e ordem de chegada respectivamente.

Estrutura do programa:
O programa feito foi dividido em 3 módulos diferentes, sendo um principal, onde está o TAD, que realiza todas as ações de teste e impressão dos resultados para o usuário, e dois auxiliares, um '.c' para a definição de todas as funções que o programa requer, e um '.h', que contém todas as chamadas das funções a serem executadas pelo arquivo principal.

lst_cria
Função responsável por criar a lista, e como será preenchida "de trás para frente", adicionamos primeiro o último elemento dela, o NULL.

```
Pacientes* lst_cria (void){
  return NULL;                    // retornar a lista vazia
}
```

lst_remove
É responsável por retirar os elementos da lista que forem atendidos.

```
Pacientes* lst_remove(Pacientes* lst,int ord){
 Pacientes*ant=NULL;              // ponteiro para o elemento anterior
 Pacientes*p=lst;                 // ponteiro para percorrer a lista
 while(p!=NULL&&p->ord!=ord){     // loop para percorrer a lista até chegar no elemento a ser removido, ou o
                                     fim da lista
   ant=p;
   p=p->prox;
 }

 if (p==NULL){                    // se não encontrar o elemento a ser removido
   return lst;
 }

 if (ant==NULL){                  // se o elemento a ser retirado for o primeiro da lista
   lst=p->prox;
 }

 else{                            // se o elemento a ser retirado estiver no meio da lista
   ant->prox=p->prox;
 }
 free(p);                         // liberar a memória alocada pelo elemento a ser removido, assim o excluindo
 return lst;
```

lst_insere_ordenado

Função que vai inserir na lista cada elemento em ordem, de acordo com as preferências anteriormente listadas.

```
Pacientes* lst_insere_ordenado (Pacientes* lst, int cor, int* ord){
  Pacientes* novo;              // ponteiro para elemento a ser inserido
  Pacientes* ant = NULL;        // ponteiro para elemento anterior
  Pacientes* p = lst;           // ponteiro para percorrer a lista

  // buscar o ponto de inserção desejado, ou o fim da lista
  while (p != NULL && p->cor <= cor && p->ord < *ord) {
  ant = p;
  p = p->prox;
  }

  // criar o novo elemento a ser inserido
  novo = (Pacientes*) malloc(sizeof(Pacientes));
  novo->cor = cor;
  novo->ord = *ord;

  // casos para a inserção do elemento
  if (ant == NULL) {            // inserir no início da lista
    novo->prox = lst;
    lst = novo;
  }

  else {                        // inserir no meio da lista
    novo->prox = ant->prox;
    ant->prox = novo;
  }

  (*ord)++;                     // aumenta o contador da ordem de chegada para os próximos que chegarem no
hospital
  return lst;                   // retorna ponteiro para o primeiro elemento
}
```

lst_imprime

Função que vai imprimir todos os elementos da lista, mostrando a ordem de chegada e a urgência correspondente.

```
void lst_imprime (Pacientes* lst){
  Pacientes* p;                             // ponteiro para percorrer a lista

  for (p = lst; p != NULL; p = p->prox){    // percorrer por cada elemento até o final da lista
    if (p->cor==1)                          // como imprimir caso a urgência seja vermelha
      printf("Ordem de chegada: %d - Cor: %s\n",p->ord, "Vermelho");
    else if (p->cor==2)                     // como imprimir caso a urgência seja amarela
      printf("Ordem de chegada: %d - Cor: %s\n",p->ord, "Amarelo");
    else                                    // como imprimir caso a urgência seja verde
      printf("Ordem de chegada: %d - Cor: %s\n",p->ord, "Verde");
  }
  printf("Fim da lista\n");                 // indicar o fim da lista
}
```

lst_libera

Responsável por liberar toda a lista, que está alocada dinamicamente.

```
void lst_libera (Pacientes*lst){
  Pacientes* p=lst;                 // ponteiro para percorrer a lista

  while (p!=NULL){                  // enquanto não chegar no final da lista
    Pacientes*t=p->prox;            // ponteiro para armazenar o endereço do próximo elemento
    free(p);                        // libera o elemento
    p=t;                            // pega o proximo elemento
  }
}
```

lst_conta_cor

É responsável por contar quantos elementos da lista estão sob cada grau de urgência, além de imprimir o resultado.

```c
void lst_conta_cor (Pacientes* lst){
  int verm=0,verd=0,amar=0;          // contadores para cada grau de urgência
  Pacientes*p;                       // ponteiro para percorrer a lista

  for(p=lst;p!=NULL;p=p->prox){      // percorrer por cada elemento até o final da lista
    if (p->cor==1)                   // caso o elemento tenha grau vermelho de urgência
      verm++;
    else if (p->cor==2)              // caso o elemento tenha grau amarelo de urgência
      amar++;
    else                             // caso o elemento tenha grau verde de urgência
      verd++;
  }

  // exibir o resultado encontrado
  printf("Numero de pacientes por cor:\nVermelho: %d - Amarelo: %d - Verde: %d\n\n",verm,amar,verd);
  return;
}
```

Solução:

A solução encontrada foi o TAD que podemos ver a seguir:

```c
int main(void) {
  Pacientes* lst=lst_cria();

  // criar o contador da ordem de chegada dos pacientes
  int* cont=(int*)malloc(sizeof(int));
  if (cont==NULL){
    printf("erro ao alocar memoria\n");
    return 1;
  }
  *cont=1;

  // inserir os pacientes de acordo com a chegada e sua urgência
  lst=lst_insere_ordenado(lst, Verde, cont);
  lst=lst_insere_ordenado(lst, Vermelho, cont);
  lst=lst_insere_ordenado(lst, Verde, cont);
  lst=lst_insere_ordenado(lst, Amarelo, cont);
  lst=lst_insere_ordenado(lst, Vermelho, cont);
  lst=lst_insere_ordenado(lst, Vermelho, cont);
  lst=lst_insere_ordenado(lst, Verde, cont);
  lst=lst_insere_ordenado(lst, Vermelho, cont);
  printf("\nChegada dos primeiros pacientes:\n");
  lst_imprime(lst);
  lst_conta_cor(lst);

  // alterações na lista
  lst=lst_remove(lst, 5);
  printf("\nPaciente 5 é atendido:\n");
  lst_imprime(lst);
  lst_conta_cor(lst);
```

1

```c
  lst=lst_remove(lst, 4);
  printf("\nPaciente 4 é atendido:\n");
  lst_imprime(lst);
  lst_conta_cor(lst);

  lst=lst_insere_ordenado(lst, Verde, cont);
  lst=lst_insere_ordenado(lst, Amarelo, cont);
  lst=lst_insere_ordenado(lst, Vermelho, cont);
  lst=lst_insere_ordenado(lst, Amarelo, cont);
  printf("\nChegam os pacientes 9, 10, 11 e 12:\n");
  lst_imprime(lst);
  lst_conta_cor(lst);

  lst=lst_remove(lst, 2);
  lst=lst_remove(lst, 6);
  printf("\nSao atendidos os pacientes 2 e 6:\n");
  lst_imprime(lst);
  lst_conta_cor(lst);

  lst=lst_remove(lst, 3);
  printf("\nPaciente 3 desiste e vai embora:\n");
  // estado final da lista
  lst_imprime(lst);
  lst_conta_cor(lst);

  lst_libera(lst);
  free(cont);

  return 0;
}
```

2

As etapas para sua criação se baseiam no método básico de uso de listas encadeadas, que é a criação dela, a inserção dos itens, e mais alterações no fim. Além disso, tivemos um cuidado especial com a formatação do texto de saída, de forma a ficar facilmente interpretável, com descrição dos valores impressos, além de dados a mais que são relevantes para a compreensão da operação.

```
Chegada dos primeiros pacientes:
Ordem de chegada: 2 - Cor: Vermelho
Ordem de chegada: 5 - Cor: Vermelho
Ordem de chegada: 6 - Cor: Vermelho
Ordem de chegada: 8 - Cor: Vermelho
Ordem de chegada: 4 - Cor: Amarelo
Ordem de chegada: 1 - Cor: Verde
Ordem de chegada: 3 - Cor: Verde
Ordem de chegada: 7 - Cor: Verde
Fim da lista
Numero de pacientes por cor:
Vermelho: 4 - Amarelo: 1 - Verde: 3


Paciente 5 é atendido:
Ordem de chegada: 2 - Cor: Vermelho
Ordem de chegada: 6 - Cor: Vermelho
Ordem de chegada: 8 - Cor: Vermelho
Ordem de chegada: 4 - Cor: Amarelo
Ordem de chegada: 1 - Cor: Verde
Ordem de chegada: 3 - Cor: Verde
Ordem de chegada: 7 - Cor: Verde
Fim da lista
Numero de pacientes por cor:
Vermelho: 3 - Amarelo: 1 - Verde: 3


Paciente 4 é atendido:
Ordem de chegada: 2 - Cor: Vermelho
Ordem de chegada: 6 - Cor: Vermelho
Ordem de chegada: 8 - Cor: Vermelho
Ordem de chegada: 1 - Cor: Verde
Ordem de chegada: 3 - Cor: Verde
Ordem de chegada: 7 - Cor: Verde
Fim da lista
Numero de pacientes por cor:
Vermelho: 3 - Amarelo: 0 - Verde: 3
```

1

```
Chegam os pacientes 9, 10, 11 e 12:
Ordem de chegada: 2 - Cor: Vermelho
Ordem de chegada: 6 - Cor: Vermelho
Ordem de chegada: 8 - Cor: Vermelho
Ordem de chegada: 11 - Cor: Vermelho
Ordem de chegada: 10 - Cor: Amarelo
Ordem de chegada: 12 - Cor: Amarelo
Ordem de chegada: 1 - Cor: Verde
Ordem de chegada: 3 - Cor: Verde
Ordem de chegada: 7 - Cor: Verde
Ordem de chegada: 9 - Cor: Verde
Fim da lista
Numero de pacientes por cor:
Vermelho: 4 - Amarelo: 2 - Verde: 4


Sao atendidos os pacientes 2 e 6:
Ordem de chegada: 8 - Cor: Vermelho
Ordem de chegada: 11 - Cor: Vermelho
Ordem de chegada: 10 - Cor: Amarelo
Ordem de chegada: 12 - Cor: Amarelo
Ordem de chegada: 1 - Cor: Verde
Ordem de chegada: 3 - Cor: Verde
Ordem de chegada: 7 - Cor: Verde
Ordem de chegada: 9 - Cor: Verde
Fim da lista
Numero de pacientes por cor:
Vermelho: 2 - Amarelo: 2 - Verde: 4


Paciente 3 desiste e vai embora:
Ordem de chegada: 8 - Cor: Vermelho
Ordem de chegada: 11 - Cor: Vermelho
Ordem de chegada: 10 - Cor: Amarelo
Ordem de chegada: 12 - Cor: Amarelo
Ordem de chegada: 1 - Cor: Verde
Ordem de chegada: 7 - Cor: Verde
Ordem de chegada: 9 - Cor: Verde
Fim da lista
Numero de pacientes por cor:
Vermelho: 2 - Amarelo: 2 - Verde: 3
```

2

Observações e conclusões:

Durante a realização deste trabalho, pudemos relembrar os conceitos aprendidos no período passado e que são de extrema relevância para o curso. Como ficamos cerca de 3 meses afastados do conteúdo, alguns detalhes básicos foram uma dificuldade no começo, principalmente em relação a características da linguagem, uma vez que a lógica e o conceito de desenvolvimento foi mais tranquilo de se conceber.Conseguimos fazer um programa completamente funcional de acordo com o solicitado, e ainda possui oportunidade de ser aprimorado conforme outras ideias se tornem requisitos.