Equipe Indecisos

Questão final - parte dois

Cada cabeçalho em negrito representa o início de um arquivo e seu nome. O código, como solicitado pela questão, começa na segunda página. Os primeiros dois arquivos são, respectivamente: instruções de compilação para o *gcc* e o *script* mencionado na última seção da parte um.

makefile

```
plataforma: energia.c bombas.c guindastes.c
gcc -o plataforma energia.c bombas.c guindastes.c -w -O2 -I.
```

custo.py

```
def p_parque(h):
    if 7 <= h < 22:
      return 3800
    return 3325

def demanda(h):
    if (6 <= h < 14) or (18 <= h):
      return 8510
    return 4620

preco = 0.503

custo_diario = 0
for h in range(24):
    custo_diario += demanda(h) - p_parque(h)
custo_diario *= preco

print("Condições ideais (operação contínua):")
print(f"Custo diário: {custo_diario:.3f}")
print(f"Custo mensal: {custo_diario*30:.3f}")
```

energia.c

```c
/** Emite comandos para os módulos de controle das bombas e dos
 *  guindastes. Faz isso baseado em cálculos de consumo de energia.
 *  Conta com um modo interativo, que permite simular situações reais
 *  de ativação e desativação de componentes, carregamento de navios
 *  e o modo de emergência das bombas.
 *  O modo interativo pode ser acesso abrindo o programa sem nenhum
 *  argumento ou com o argumento opcional -t, que permite escolher um
 *  horário inicial diferente do padrão (12:00).
 *  Além disso, pode ser aberto no modo custo, que calcula o custo
 *  diário e mensal de operação da plataforma, de acordo com as
 *  especificaçes do desafio.
 */

#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>

#include "energia.h"

int main(int argc, char **argv)
{
     // Variáveis que dependem dos argumentos.
     int hora = 12, minuto = 0, segundo = 0;
     /* ------------------------------------------------------------
     ---------------- ARGUMENTOS ----------------------------------
     ------------------------------------------------------------ */
     // Se o programa for aberto com nenhum argumento, entra no modo
     // interativo com o horário padrão (12:00).
     if (argc == 1)
     {
     // Modo interativo padrão.
     }
     // Se o programa for aberto com 1 argumento:
     else if (argc == 2)
     {
```

```c
    // Modo simulação: calcula o gasto de energia diário e mensal
    // (30 dias), em situações ideais.
    if (!strcmp(argv[1], "custo"))
    {
        hora = 0, minuto = 0;
        // Cria uma plataforma padrão, com 25 séries de bombas e
        // 10 guindastes.
        Bombas *bombas = CriarBombas(NUM_BOMBAS);
        Guindastes *guindastes = CriarGuindastes(NUM_GUINDASTES);
        if (bombas == NULL || guindastes == NULL)
        {
            // Remove as bombas e guindastes da memória.
            removerBombeamento(bombas);
            removerGuindastes(guindastes);
            return 2;
        }
        // Cria um navio com capacidade extrema, simulando uma
        // situação em que a troca de navios é instantânea.
        atualizarNavio(guindastes, INT_MAX);
        // Dá 60*24*60*30 passos (1 mês), registrando o custo
        // durante o processo.
        double custoMensal = passosN(60 * 60 * 24 * 30, bombas,
guindastes,
                                     &hora, &minuto, &segundo, false);
        // Mostra os custos calculados no terminal.
        printf("Condições ideais (operação contínua):\n");
        printf("Custo diário: R$ %.3lf\n", custoMensal / 30);
        printf("Custo mensal: R$ %.3lf\n", custoMensal);
        // Remove as bombas e guindastes da memória.
        removerBombeamento(bombas);
        removerGuindastes(guindastes);
        return 0;
    }
    // Ajuda: mostra os comandos possíveis no terminal.
    else if (!strcmp(argv[1], "-h") || !strcmp(argv[1], "--help"))
    {
        ajudaDoPrograma();
        return 0;
    }
    // Comando não identificado: instruções básicas.
```

```
    else
    {
          printf("Use 'plataforma --help' para obter ajuda.\n");
          return 1;
    }
    }
    // Se o programa for aberto com 3 argumentos:
    else if (argc == 4)
    {
    if (strcmp(argv[1], "-t"))
    {
          printf("Use 'plataforma --help' para obter ajuda.\n");
          return 1;
    }
    // Comando -t: permite inserir um horário inicial qualquer
    // para o modo interativo.
    // Se alguma das entradas não for um número: instruções de
    // uso do comando -t.
    if (!strNumerica(argv[2]) || !strNumerica(argv[3]))
    {
          printf("Uso: plataforma -t horas minutos\n");
          return 1;
    }
    // Se ambas as entradas são números: os insere como horário
    // inicial, limitando os valores de hora e minuto para 0-23
    // e 0-59, respectivamente.
    char *p; // Variável não usada, necessária para strtol.
    minuto = (int)(strtol(argv[3], &p, 10) % 60);
    hora = (int)((strtol(argv[3], &p, 10) / 60
               + strtol(argv[2], &p, 10)) % 24);
    }
    // Outras quantidades de argumentos: instruções básicas.
    else
    {
    printf("Use 'plataforma --help' para obter ajuda.\n");
    return 1;
    }

    /* ------------------------------------------------------------
    -------------- MODO INTERATIVO ---------------------------------
```

```
     ------------------------------------------------------------ */

     // Cria uma plataforma padrão, com 25 séries de bombas, 10
     // guindastes e um navio com capacidade padrão.
     Bombas *bombas = CriarBombas(NUM_BOMBAS);
     Guindastes *guindastes = CriarGuindastes(NUM_GUINDASTES);
     if (bombas == NULL || guindastes == NULL)
     {
     // Remove as bombas e guindastes da memória.
     removerBombeamento(bombas);
     removerGuindastes(guindastes);
     return 2;
     }
     atualizarNavio(guindastes, CAPACIDADE_DO_NAVIO);
     // mostrarFracao: true se o usuário quer ver o uso da termelétrica
     // a cada passo, false se não.
     bool mostrarFracao = false;

     printf("----- MODO INTERATIVO -----\n");
     printf("Digite 'h' para obter ajuda\n");
     printf("---------------------------\n\n");
     double custoTotal = 0;
     while (true)
     {
     // Mostra informações resumidas.
     printf("Horário: %02d:%02d.%02d\n", hora, minuto, segundo);
     printf("Bombas ativas: %d de %d\n", bombas->ativas,
bombas->totais);
     printf("Guindastes ativos: %d de %d\n", guindastes->ativos,
guindastes->totais);
     printf("Capacidade do navio: %d barris\n",
guindastes->estadoDoNavio);
     // Solicita e executa um comando.
     while (true)
     {
          // Solicita um char do usuário.
          printf("-> ");
          char comando;
          scanf(" %c", &comando);
          // Cria uma variável para o custo e para um int qualquer.
```

```
        double custo;
        int n;
        switch (comando)
        {
              // Comando 'P': avança a simulação em algum número de
              // passos (até um dia).
              case 'P':
              n = getNum(0, 24*60*60);
              custo = passosN(n, bombas, guindastes, &hora,
                              &minuto, &segundo, mostrarFracao);
              printf("\nCusto: R$ %.3lf\n", custo);
              custoTotal += custo;
              break;
              // Comando 'p': avança a simulação um passo.
              case 'p':
              custo = passosN(1, bombas, guindastes, &hora,
                              &minuto, &segundo, mostrarFracao);
              printf("\nCusto: R$ %.3lf\n", custo);
              custoTotal += custo;
              break;
              // Comando 'E': desativa o modo de emergência das
              // bombas.
              case 'E':
              normalizacaoDoBombeamento(bombas);
              break;
              // Comando 'e': ativa o modo de emergência das bombas.
              case 'e':
              emergenciaDoBombeamento(bombas);
              break;
              // Comando 'G': altera o número máximo de guindastes
              // ativos.
              case 'G':
              n = getNum(0, guindastes->totais);
              guindastes->ativosMax = n;
              continue;
              // Comando 'g': mostra o estado dos guindastes.
              case 'g':
              estadoDosGuindastes(guindastes);
              continue;
              // Comando 'B': altera o número de bombas ativas.
```

```
                case 'B':
                n = getNum(0, bombas->totais);
                alterarBombasAtivas(bombas, n);
                continue;
                // Comando 'b': mostra o estado das bombas.
                case 'b':
                estadoDoBombeamento(bombas);
                continue;
                // Comando 'N': avança a simulação até o navio
                // atual estar cheio.
                case 'N':
                custo = passosNavio(bombas, guindastes, &hora,
                                    &minuto, &segundo,
mostrarFracao);
                printf("\nCusto: R$ %.3lf\n", custo);
                custoTotal += custo;
                break;
                // Comando 'n': significa a chegada de um navio.
                case 'n':
                // Se não há um navio já atracado, continua.
                if(atualizarNavio(guindastes, CAPACIDADE_DO_NAVIO))
                {
                      printf("Navio atracado.\n");
                      printf("Capacidade do navio: %d barris\n",
                            guindastes->estadoDoNavio);
                      continue;
                }
                // Se há, pede um comando novo.
                else
                {
                      printf("Já há um navio atracado.\n");
                      continue;
                }
                // Comando 'T/t': decide se a demanda da termelétrica
                // será mostrada na tela durante o modo interativo.
                case 'T':
                case 't':
                mostrarFracao = !mostrarFracao;
                if (mostrarFracao)
                {
```

```
                    printf("Demanda da termelétrica será
mostrada.\n");
                }
                else
                {
                    printf("Demanda da termelétrica não será
mostrada.\n");
                }
                continue;
                // Comando 'H/h': mostra ajuda do modo interativo.
                case 'H':
                case 'h':
                ajudoDoModoInterativo();
                continue;
                // Comando 'Q/q': sai do programa.
                case 'Q':
                case 'q':
                // Remove as bombas e guindastes da memória.
                removerBombeamento(bombas);
                removerGuindastes(guindastes);
                return 0;
                // Se o comando não é reconhecido, pede um novo.
                default:
                printf("Comando inválido.\n");
                continue;
            }
            break;
    }
    printf("Custo total: R$ %.3lf\n\n", custoTotal);
    }

    // Remove as bombas e guindastes da memória.
    removerBombeamento(bombas);
    removerGuindastes(guindastes);
    return 0;
}
```

```
/* Dá uma quantidade pré-determinada de passos, e retorna o custo
total dessas etapas. */
```

```
double passosN(int passos, Bombas *bombas, Guindastes *guindastes,
                int *hora, int *minuto, int *segundo,
                bool mostrarFracao)
{
     double fracaoDaTermeletrica;
     double custo = 0;
     for (int i = 0; i < passos; i++)
     {
     passo(bombas, guindastes, hora, minuto, segundo,
          &fracaoDaTermeletrica,
          mostrarFracao);
     custo += fracaoDaTermeletrica * P_TERMELETRICA *
                C_TERMELETRICA / 3600;
     }
     return custo;
}
```

```
/* Funciona como passosN, mas em vez de dar uma quantidade pré-
-determinada de passos, avança a simulação até o navio atracado
na plataforma atingir sua capacidade. */
double passosNavio(Bombas *bombas, Guindastes *guindastes, int *hora,
                int *minuto, int *segundo, bool mostrarFracao)
{
     if (guindastes->estadoDoNavio == 0)
     {
     return 0.0;
     }
     double fracaoDaTermeletrica;
     double custo = 0;
     while (passo(bombas, guindastes, hora, minuto, segundo,
          &fracaoDaTermeletrica, mostrarFracao))
     {
     custo += fracaoDaTermeletrica * P_TERMELETRICA *
                C_TERMELETRICA / 3600;
     }
     return custo;
}
```

```
/* Simula um passo (um minuto) de operação da plataforma. Retorna
true se há um navio na plataforma, false se não. A fração da
capacidade da termelétrica que é demandada pela plataforma é
colocada no endereço de memória especificado. */
bool passo(Bombas *bombas, Guindastes *guindastes, int *hora,
           int *minuto, int *segundo, double *fracaoDaTermeletrica,
           bool mostrarFracao)
{
      // Acresce o tempo em um segundo.
      (*segundo)++;
      if (*segundo >= 60)
      {
      *segundo = 0;
      (*minuto)++;
      }
      if (*minuto >= 60)
      {
      *minuto = 0;
      (*hora)++;
      *hora %= 24;
      }
      // Avança a posição dos guindastes.
      bool estadoDoNavio = atualizarGuindastes(guindastes, *hora);
      // Calcula a distribuição de energia.
      *fracaoDaTermeletrica = ajustarDemanda(bombas, guindastes, *hora);
      // Mostra a fração da energia usada em um determinado horário.
      if (mostrarFracao)
      {
      printf("\n(%02d:%02d.%02d) %.2lf %%", *hora, *minuto, *segundo,
                  *fracaoDaTermeletrica * 100);
      }
      // Retorna o estado do navio.
      return estadoDoNavio;
}
```

```
/* Calcula a porcentagem da demanda da termelétrica que deve ser
direcionada para a plataforma de petróleo para a sua operação e, se
necessário, ajusta a quantidade de guindastes que podem ser
ativados. Retorna a fração da potência fornecida pela termelétrica
```

```
que deve ser direcionada à plataforma. */
double ajustarDemanda(Bombas *bombas, Guindastes *guindastes, int
horario)
{
      // Primeiro, calcula a demanda total de energia no momento.
      double demandaTotal = P_AUXILIAR;
      demandaTotal += bombas->ativas * P_BOMBA;
      demandaTotal += guindastes->ativos * P_GUINDASTE;
      // Então, calcula quanta dessa energia deve vir da termelétrica.
      double subdemanda = demandaDaTermeletrica(demandaTotal, horario);
      // Se a demanda da termelétrica é menor que a sua capacidade de
      // fornecimente, desliga primeiro os guindastes e depois as
      // bombas, até que a energia demandada possa ser fornecida pela
      // usina.
      while (subdemanda > P_TERMELETRICA && guindastes->ativos > 0)
      {
      guindastes->ativos--;
      subdemanda -= P_GUINDASTE / E_INVERSORES;
      }
      while (subdemanda > P_TERMELETRICA && bombas->ativas > 0)
      {
      alterarBombasAtivas(bombas, bombas->ativas - 1);
      subdemanda -= P_BOMBA / E_INVERSORES;
      }
      // Se a demanda da termelétrica ainda e maior que ela é capaz de
      // fornecer, solicita toda a potência da usina.
      if (subdemanda > P_TERMELETRICA)
      {
      return 1.0;
      }
      // Se não, calcula a fração da capacidade da usina que deve ser
      // direcionada à plataforma.
      /** Em um sistema real, uma interface programa -> dispositivo
      seria utilizada para ajustar a demanda de energia de acordo com
      esse valor. */
      return subdemanda / P_TERMELETRICA;
}
```

```
/* Calcula a potência que deve ser fornecida pela termelétrica, dado
```

```c
um horário do dia e uma demanda total, em kW. */
double demandaDaTermeletrica(double demandaTotal, int horario)
{
	double demanda = demandaTotal - potenciaDasTurbinas(horario);
	// Se a potência das turbinas é suficiente para suprir a demanda.
	if (demanda < 0)
	{
	return 0;
	}
	return demanda / E_INVERSORES;
}
```

```c
/* Calcula a potência gerada pelas turbinas eólicas, em kW, em um
certo horário do dia. */
double potenciaDasTurbinas(int horario)
{
	if (horario > 7 && horario < 22)
	{
	// 80 kW = potência quando v = 6 m/s.
	return 80 * NUM_TURBINAS * E_INVERSORES;
	}
	// 70 kW = potência quando v = 10 m/s.
	return 70 * NUM_TURBINAS * E_INVERSORES;
}
```

```c
/* Solicita um número do usuário dentro de um limite. */
int getNum(int minimo, int maximo)
{
	int numero;
	printf("Entre um número entre %d e %d:\n", minimo, maximo);
	do
	{
	printf("-> ");
	scanf("%d", &numero);
	}
	while (numero < minimo || numero > maximo);
	return numero;
}
```

```c
/* Retorna true se str for uma string numérica, false se não. */
bool strNumerica(char *str)
{
	for (int i = 0; str[i] != '\0'; i++)
	{
	if (!isdigit(str[i]))
	{
		return false;
	}
	}
	return true;
}
```

```c
/* Mostra as instruções de uso do programa e do modo interativo,
respectivamente. */
void ajudaDoPrograma(void)
{
	printf("Uso:\n");
	// Modo de uso: interativo.
	printf("\tplataforma [opções]\n");
	printf("\tAbre o programa no modo interativo.\n\n");
	// Modo de uso: custo.
	printf("\tplataforma custo\n");
	printf("\tAbre o programa no modo custo.\n\n");
	// Opções.
	printf("\tOpções:\n");
	printf("\t\t-h --help\n\t\t\tExibe este menu de ajuda\n");
	printf("\t\t-t [hora] [minuto]\n\t\t\tAltera o horário inicial ");
	printf("do modo interativo.\n");
}
void ajudoDoModoInterativo(void)
{
	printf("Os comandos que AVANÇAM a simulação mostram o custo do
avanço");
	printf(" atual, o custo total desde o início do programa, e um
resumo");
	printf(" do estado da simulação.\n");
```

```
    printf("COMANDOS:\n");
    printf("b : exibe o estado de todo o sistema de bombeamento.\n");
    printf("B : permite alterar o número de bombas ativas.\n");
    printf("e : ativa o estado de emergência das bombas.\n");
    printf("E : desativa o esta do emergência das bombas.\n");
    printf("g : exibe o estado de todo o sistema de guindastes.\n");
    printf("G : permite alterar o número máximo de guindastes
ativos.\n");
    printf("h : exibe este menu de ajuda.\n");
    printf("H : exibe este menu de ajuda.\n");
    printf("n : tenta atracar um navio.\n");
    printf("N : AVANÇA a simulação até o navio atracado estiver
cheio.\n");
    printf("p : AVANÇA a simulação um passo.\n");
    printf("P : AVANÇA a simulação um número arbitrário de
passos.\n");
    printf("q : fecha o programa.\n");
    printf("Q : fecha o programa.\n");
    printf("t : decide se a demanda da termelétrica em cada horário
será mostrada.\n");
    printf("T : decide se a demanda da termelétrica em cada horário
será mostrada.\n");
}
```

energia.h

```
#ifndef _ENERGIA
#define _ENERGIA

#include <stdbool.h>

#include "bombas.h"
#include "guindastes.h"

/* Número de turbinas eólicas, definido pelo desafio. */
#define NUM_TURBINAS 50
/* Potência dos sistemas auxiliares da plataforma, em kW, definida
pelo desafio. */
#define P_AUXILIAR 3620
/* Potência máxima que pode ser fornecida pela termelétrica, em kW,
calculada pela equipe baseado nos dados fornecidos pelo desafio. */
```

```c
#define P_TERMELETRICA 231008
/* Custo em reais por kWh da energia fornecida pela termelétrica,
calculado pela equipe baseado nos dados fornecidos pelo desafio. */
#define C_TERMELETRICA 0.478
/* Eficiência dos inversores de frequência, definida pelo desafio. */
#define E_INVERSORES 0.95

/* Dá uma quantidade pré-determinada de passos, e retorna o custo
total dessas etapas. */
double passosN(int passos, Bombas *bombas, Guindastes *guindastes,
               int *hora, int *minuto, int *segundo, bool
mostraFracao);

/* Funciona como passosN, mas em vez de dar uma quantidade pré-
-determinada de passos, avança a simulação até o navio atracado
na plataforma atingir sua capacidade. */
double passosNavio(Bombas *bombas, Guindastes *guindastes, int *hora,
               int *minuto, int *segundo, bool mostrarFracao);

/* Simula um passo (um minuto) de operação da plataforma. Retorna
true se há um navio na plataforma, false se não. A fração da
capacidade da termelétrica que é demandada pela plataforma é
colocada no endereço de memória especificado. */
bool passo(Bombas *bombas, Guindastes *guindastes, int *hora,
           int *minuto, int *segundo, double *fracaoDaTermeletrica,
           bool mostrarFracao);

/* Calcula a porcentagem da demanda da termelétrica que deve ser
direcionada para a plataforma de petróleo para a sua operação e, se
necessário, ajusta a quantidade de guindastes que podem ser
ativados. Retorna a fração da potência fornecida pela termelétrica
que deve ser direcionada à plataforma. */
double ajustarDemanda(Bombas *bombas, Guindastes *guindastes,
                      int horario);

/* Calcula a potência que deve ser fornecida pela termelétrica, dado
um horário do dia e uma demanda total, em kW. */
double demandaDaTermeletrica(double demandaTotal, int horario);

/* Calcula a potência gerada pelas turbinas eólicas, em kW, em um
```

```
certo horário do dia. */
double potenciaDasTurbinas(int horario);

/* Solicita um número do usuário dentro de um limite. */
int getNum(int minimo, int maximo);

/* Retorna true se str for uma string numérica, false se não. */
bool strNumerica(char *str);

/* Mostra as instruções de uso do programa e do modo interativo,
respectivamente. */
void ajudaDoPrograma(void);
void ajudoDoModoInterativo(void);

#endif // _ENERGIA
```

bombas.c

```c
/** Controla as bombas, de acordo com as especificaçes do desafio.
 *  Além disso, controla as luzes indicadoras de operação e de
 *  emergência.
 *  Em seu estado padrão, todas as bombas estão operando, e a luz
 *  amarela está acesa.
 *  Caso o botão de emergência seja pressionado, as bombas são
 *  desligadas, a luz vermelha é acessa, e a luz amarela é desligada.
 *  Ao receber comandos do controlador principal, o número de bombas
 *  ativas pode mudar, e a luz amarela é alterada de acordo.
 */

#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>

#include "bombas.h"

/* Cria e inicializa um sistema de bombeamento. No início, todas as
bombas estão ativas. */
Bombas *CriarBombas(int num_bombas)
{
     // Se o número de bombas for inválido (menor que 1) retorna
     // um apontador nulo.
     if (num_bombas < 1)
     {
     return NULL;
     }
     // Tenta reservar espaço para um sistema de bombeamento. Se isso
     // falhar, retorn um apontador nulo.
     Bombas *bombas = malloc(sizeof(Bombas));
     if (bombas == NULL)
     {
     return NULL;
     }
     // Tenta criar uma lista para guardar os estados. Se isso falhar,
     // retorna um apontador nulo.
     bombas->estados = malloc(num_bombas * sizeof(bool));
     if (bombas->estados == NULL)
```

```
    {
    return NULL;
    }
    for (int i = 0; i < num_bombas; i++)
    {
    bombas->estados[i] = true;
    }
    // Inicializa o restante das variáveis.
    bombas->totais = num_bombas;
    bombas->ativas = num_bombas;
    bombas->luzAmarela = true;
    bombas->luzVermelha = false;
    return bombas;
}
```

```
/* Mostra ATIVA se o estado for verdadeiro, INATIVA se for
falso. Utilizado por estadoDoBombeamento. Função local. */
static void mostrarEstadoDaBomba(bool estado)
{
    if (estado)
    {
    printf("ATIVA");
    }
    else
    {
    printf("INATIVA");
    }
    printf("\n");
}
```

```
/* Mostra o estado de todos os componentes de um sistema de
bombeamento no terminal. */
void estadoDoBombeamento(Bombas *bombas)
{
    printf("Bombas totais: %d\n", bombas->totais);
    printf("Bombas ativas: %d\n", bombas->ativas);
    printf("  Estados das séries de bombas:\n");
    for (int i = 0; i < bombas->totais; i++)
```

```
    {
    printf("  | Série %02d: ", i+1);
    mostrarEstadoDaBomba(bombas->estados[i]);
    }
    printf("Luz amarela: ");
    mostrarEstadoDaBomba(bombas->luzAmarela);
    printf("Luz vermelha: ");
    mostrarEstadoDaBomba(bombas->luzVermelha);
}
```

```
/* Altera o número de bombas ativas de um sistema de bombeamento,
atualizando ao mesmo tempo o estado de cada série de bombas e da luz
indicadora. */
void alterarBombasAtivas(Bombas *bombas, int ativas)
{
    // Bloqueia a ativação das bombas caso o modo de emergência
    // esteja ativado.
    if (bombas->luzVermelha)
    {
    return;
    }
    bombas->ativas = ativas;
    for (int i = 0; i < bombas->totais; i++)
    {
    bombas->estados[i] = i < ativas;
    }
    bombas->luzAmarela = ativas;
}
```

```
/* Ativa o estado de emergência das bombas. */
void emergenciaDoBombeamento(Bombas *bombas)
{
    alterarBombasAtivas(bombas, 0);
    bombas->luzVermelha = true;
}
```

```
/* Desativa o estado de emergência das bombas. */
```

```c
void normalizacaoDoBombeamento(Bombas *bombas)
{
      bombas->luzVermelha = false;
}
```

```c
/* Remove o sistema de bombeamento da memória. */
void removerBombeamento(Bombas *bombas)
{
      if (bombas != NULL)
      {
      free(bombas->estados);
      }
      free(bombas);
}
```

bombas.h

```c
#ifndef _BOMBAS
#define _BOMBAS

#include <stdbool.h>

/* Número de séries de bombas, definido pelo desafio. */
#define NUM_BOMBAS 25
/* Potência das séries de bombas, definida pelo desafio, em kW por
série de bombas ativa. */
#define P_BOMBA 40

/** Representação programática de um sistema de bombeamento. Em um
sistema real, os valores das variáveis estados (das bombas), luzAmarela
e luzVermelha seriam usados para controlar os respectivos dispositivos
mecânicos, através de uma interface controlador -> dispositivo. */
typedef struct {
      // Número de bombas totais.
      int totais;
      // Número de bombas ativas.
      int ativas;
      // Nos itens seguintes, true = ativa, false = inativa.
      // Lista que representa o estado de cada bomba.
      bool *estados;
      // Ativada quando ativas > 0.
      bool luzAmarela;
      // Ativada quando o botão é pressionado.
      bool luzVermelha;
} Bombas;

/** Protótipos das funções públicas, utilizadas pelo controlador
principal. */

/* Cria e inicializa um sistema de bombeamento. No início, todas as
bombas estão ativas. */
Bombas *CriarBombas(int num_bombas);

/* Mostra o estado de todos os componentes de um sistema de
bombeamento no terminal. */
```

```c
void estadoDoBombeamento(Bombas *bombas);

/* Altera o número de bombas ativas de um sistema de bombeamento,
atualizando ao mesmo tempo o estado de cada série de bombas e da luz
indicadora. */
void alterarBombasAtivas(Bombas *bombas, int ativas);

/* Ativa o estado de emergência das bombas. */
void emergenciaDoBombeamento(Bombas *bombas);

/* Desativa o estado de emergência das bombas. */
void normalizacaoDoBombeamento(Bombas *bombas);

/* Remove o sistema de bombeamento da memória. */
void removerBombeamento(Bombas *bombas);

#endif // _BOMBAS
```

guindastes.c

```c
/** Controla os guindastes, de acordo com as especificaçes do desafio.
 *  A ativação e desativação dos guindastes é feita de modo a carregar
 *  o navio o mais rápido possível, respeitando os limites impostos
 *  pelo controlador de energia.
 *  Em seu estado padrão, todas os guindastes estão operando.
 *  Quando fora do horário de funcionamento (06:00 - 14:00 e 18:00 -
 *  00:00), ou quando não há um navio atracado, os guindastes são
 *  parados.
 */

#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>

#include "guindastes.h"

/* Cria e inicializa um grupo de guindastes. No início, todos os
guindastes estão inativos, mas prontos para carregar um barril. */
Guindastes *CriarGuindastes(int num_guindastes)
{
     // Se o número de guindastes for inválido (menor que 1) retorna
     // um apontador nulo.
     if (num_guindastes < 1)
     {
     return NULL;
     }
     // Tenta reservar espaço para um grupo de guindastes. Se isso
     // falhar, retorn um apontador nulo.
     Guindastes *guindastes = malloc(sizeof(Guindastes));
     if (guindastes == NULL)
     {
     return NULL;
     }
     // Tenta criar listas para guardar os estados e os progressos.
     // Se isso falhar, retorna um apontador nulo.
     guindastes->progressos = malloc(num_guindastes * sizeof(int));
     guindastes->estados = malloc(num_guindastes * sizeof(bool));
     if (guindastes->progressos == NULL || guindastes->estados == NULL)
```

```
    {
    return NULL;
    }
    for (int i = 0; i < num_guindastes; i++)
    {
    guindastes->progressos[i] = -TEMPO_DE_COLETA;
    guindastes->estados[i] = false;
    }
    // Inicializa o restante das variáveis.
    guindastes->totais = num_guindastes;
    guindastes->ativos = 0;
    guindastes->ativosMax = num_guindastes;
    guindastes->carregando = 0;
    guindastes->estadoDoNavio = 0;
    return guindastes;
}
```

```
/* Mostra ATIVO se o estado for verdadeiro, INATIVO se for
falso. Utilizado por estadoDosGuindastes. Função local. */
static void mostrarEstadoDoGuindaste(bool estado)
{
    if (estado)
    {
    printf("ATIVO");
    }
    else
    {
    printf("INATIVO");
    }
}
```

```
/* Mostra o estado de todos os componentes de um grupo de guindastes
no terminal. */
void estadoDosGuindastes(Guindastes *guindastes)
{
    printf("Guindastes totais: %d\n", guindastes->totais);
    printf("Guindastes ativos: %d\n", guindastes->ativos);
    printf("Guindastes ativos (max): %d\n", guindastes->ativosMax);
```

```
    printf("Guindastes carregando: %d\n", guindastes->carregando);
    printf("  Guindastes: estado (progresso)\n");
    for (int i = 0; i < guindastes->totais; i++)
    {
    printf("  | Guindaste %02d: ", i+1);
    mostrarEstadoDoGuindaste(guindastes->estados[i]);
    printf(" (%02d)\n", guindastes->progressos[i]);
    }
    printf("Estado do navio: %d\n", guindastes->estadoDoNavio);
}
```

```
/* Desativa todos os guindastes. Função local. */
static void desativarTodosOsGuindastes(Guindastes *guindastes)
{
    guindastes->ativos = 0;
    for (int i = 0; i < guindastes->totais; i ++)
    {
    guindastes->estados[i] = false;
    }
}
```

```
/* Altera o número máximo de guindastes ativos de um grupo de
guindastes, atualizando ao mesmo tempo o estado de cada guindaste.
Tem preferência por desativar guindastes com menor progresso, e
reativar guindastes com maior progresso, economizando energia a longo
prazo. Função local. */
static void alterarGuindastesAtivos(Guindastes *guindastes)
{
    // Primeiro, desativa todos os guindastes.
    desativarTodosOsGuindastes(guindastes);
    // Então, reativa guindaste até o número de guindastes ativos
    // ser igual ao número de guindastes ativos máximo.
    while (guindastes->ativos != guindastes->ativosMax)
    {
    guindastes->ativos++;
    int maiorIndice, maiorProgresso;
    // Acha um guindaste inativo e guarda seu índice e progresso.
    for (int i = 0; i < guindastes->totais; i++)
```

```
	{
		if (!guindastes->estados[i])
		{
			maiorIndice = i;
			maiorProgresso = guindastes->progressos[i];
			break;
		}
	}
	// Então, tenta achar o guindaste inativo com maior progresso.
	for (int i = maiorIndice + 1; i < guindastes->totais; i++)
	{
		if (!guindastes->estados[i]
			&& guindastes->progressos[i] > maiorProgresso)
		{
			maiorIndice = i;
			maiorProgresso = guindastes->progressos[i];
		}
	}
	// Finalmente, ativa o guindaste com maior progresso encontrado.
	guindastes->estados[maiorIndice] = true;
	}
}
```

```
/* Avança o estado de todos os componentes do grupo de guindastes
em um minuto. A alteração dos estados depende do horário, já que os
guindastes não funcionam 24 h por dia. O horário é dado em horas. */
bool atualizarGuindastes(Guindastes *guindastes, int horario)
{
	// Se o horário estiver fora dos horários de funcionamento dos
	// guindastes ou não houver um navio atracado, desativa todos
	// eles.
	if (guindastes->estadoDoNavio == 0
	|| horario < 6
	|| (horario >= 14 && horario < 18)
	|| horario >= 24)
	{
	desativarTodosOsGuindastes(guindastes);
	return guindastes->estadoDoNavio != 0;
	}
```

```
    // Se o horário estiver dentro dos horários de funcionamento dos
    // guindastes e houver um navio atracado, atualiza o número de
    // guindastes ativos, para que seja igual ao número máximo de
    // guindastes ativos.
    alterarGuindastesAtivos(guindastes);
    // Atualiza a posição de cada guindaste.
    for (int i = 0; i < guindastes->totais; i++)
    {
    // Se um guindaste está inativo, ele continua parado.
    if (!guindastes->estados[i])
    {
          continue;
    }
    // Se o guindaste tiver chegado na posição original, decide
    // se vai carregar um barril ou ficar parado.
    if (guindastes->progressos[i] == 0)
    {
          // Se o número de guindastes carregando barris for o
          // suficiente para encher o barco, o guindaste fica
          // parado.
          if (guindastes->carregando >= guindastes->estadoDoNavio)
          {
                guindastes->ativos--;
                guindastes->estados[i] = false;
                continue;
          }
          // Se não, o guindaste começa a carregar um barril.
          else
          {
                guindastes->carregando++;
          }
    }
    // Se o guindaste tiver terminado de carregar um barril,
    // diminui o número de guindastes carregando barris e faz
    // eles começar a coletar outro.
    else if (guindastes->progressos[i] == TEMPO_DE_CARREGAMENTO - 1)
    {
          guindastes->carregando--;
          guindastes->progressos[i] = -1 - TEMPO_DE_COLETA;
          guindastes->estadoDoNavio--;
```

```
        }
        // A posição do guindaste é avançanda em um passo.
        /** Em um sistema real, a posição do guindaste poderia ser
        determinada empiricamente, em vez de ser simulada. */
        guindastes->progressos[i]++;
        }
        return true;
}
```

```
/* Atualiza o valor da capacidade do navio de um grupo de guindastes
quando um novo navio é atracado. A capacidade é um valor inteiro,
e representa a quantidade de barris que o novo navio ainda pode
comportar. */
bool atualizarNavio(Guindastes *guindastes, int capacidade)
{
        if (guindastes->estadoDoNavio <= 0)
        {
        guindastes->estadoDoNavio = capacidade;
        return true;
        }
        return false;
}
```

```
/* Remove o grupo de guindastes da memória. */
void removerGuindastes(Guindastes *guindastes)
{
        if (guindastes != NULL)
        {
        free(guindastes->progressos);
        free(guindastes->estados);
        }
        free(guindastes);
}
```

guindastes.h

```
#ifndef _GUINDASTES
```

```c
#define _GUINDASTES

#include <stdbool.h>

/* Número de guindastes, definido pelo desafio. */
#define NUM_GUINDASTES 10
/* Potência de cada guindaste, definida pela equipe, em kW por
guindaste ativo. */
#define P_GUINDASTE 389

/* Tempo necessário para um guindaste coletar um novo barril após
ter carregado outro barril, em segundos. Definido pela equipe */
#define TEMPO_DE_COLETA 13
/* Tempo necessário para um guindaste carregar um barril no navio após
ele ser coletado, em segundos. Definido pela equipe. */
#define TEMPO_DE_CARREGAMENTO 17
/* Capacidade do navio, determinado pela equipe. */
#define CAPACIDADE_DO_NAVIO 203349

/** Representação programática de um grupo de guindastes. Em um
sistema real, os valores da variável estados (dos guindastes) seriam
usados para controlar os respectivos dispositivos mecânicos, através
de uma interface controlador -> dispositivo. */
typedef struct {
      // Número de guindastes totais.
      int totais;
      // Número de guindastes ativos.
      int ativos;
      // Número máximo de guindastes ativos, definido pelo suprimento de
      // energia.
      int ativosMax;
      // Número de guindastes que estão, no momento, carregando um
      // barril no navio.
      int carregando;
      // Número que representa o estado do navio atracado junto aos
      // guindastes. Quando igual a zero, não há um navio atracado.
      // Quando positivo, indica o número de barris que o navio atracado
      // ainda pode comportar.
      int estadoDoNavio;
      // Número que representa o progresso de um determinado guindaste,
```

```
    // no processo de carregamento de um barril ou coleta de um novo
    // barril. Quando negativo, o guindaste está procurando um novo
    // barril; quando positivo, o guindaste está carregando um barril
    // no navio.
    int *progressos;
    // Lista que representa o estado de cada guindaste, onde
    // true = ativo, false = inativo.
    bool *estados;
} Guindastes;

/** Protótipos das funções públicas, utilizadas pelo controlador
principal. */

/* Cria e inicializa um grupo de guindastes. No início, todos os
guindastes estão inativos e prontos para carregar um barril. */
Guindastes *CriarGuindastes(int num_guindastes);

/* Mostra o estado de todos os componentes de um grupo de guindastes
no terminal. */
void estadoDosGuindastes(Guindastes *guindastes);

/* Avança o estado de todos os componentes do grupo de guindastes
em um minuto. A alteração dos estados depende do horário, já que os
guindastes não funcionam 24 h por dia. O horário é dado em horas.
Retorna true se o navio atracado ainda tiver capacidade após o
carregamento, false se não houver um navio atracado ou se o navio
atracado estiver cheio. */
bool atualizarGuindastes(Guindastes *guindastes, int horario);

/* Tenta atualizar o valor da capacidade do navio de um grupo de
guindastes quando um novo navio cehga. A capacidade é um valor
inteiro, e representa a quantidade de barris que o novo navio ainda
pode comportar. Retorna true se o novo navio pôde ser atracado, falso
se não, ou seja, quando outro navio ainda estava no porto. */
bool atualizarNavio(Guindastes *guindastes, int capacidade);

/* Remove o grupo de guindastes da memória. */
void removerGuindastes(Guindastes *guindastes);

#endif // _GUINDASTES
```