N_listas.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

// Estrutura do Nó e da Lista Encadeada
typedef struct Node {
    int data;
    struct Node* next;
} Node;

typedef struct LinkedList {
    Node* head;
} LinkedList;

// Implementação da função para inserir um novo nó no início da lista
void insert(LinkedList* list, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        // Em um programa real, um tratamento de erro mais robusto seria necessário
        return;
    }
    newNode->data = value;
    newNode->next = list->head;
    list->head = newNode;
}

// Implementação da função para liberar a memória de uma lista
void free_list(LinkedList* list) {
    Node* current = list->head;
    while (current != NULL) {
        Node* temp = current;
        current = current->next;
        free(temp);
    }
    list->head = NULL;
}

int main() {
    const int N_INSERTIONS = 100000;
    int M_LISTS;

    printf("Digite o número de listas: ");
    scanf("%d", &M_LISTS);

    if (M_LISTS <= 0) {
        printf("Número de listas deve ser positivo.\n");
        return 1;
    }
```

```
50
51      // 1. Aloca dinamicamente um array de listas
52      LinkedList* lists = (LinkedList*)malloc(M_LISTS * sizeof(LinkedList));
53      // 2. Aloca dinamicamente um array de locks
54      omp_lock_t* locks = (omp_lock_t*)malloc(M_LISTS * sizeof(omp_lock_t));
55
56      // Verificação de robustez da alocação
57      if (lists == NULL || locks == NULL) {
58          fprintf(stderr, "Falha ao alocar memória para listas ou locks.\n");
59          free(lists);
60          free(locks);
61          return 1;
62      }
63
64      // Inicializa cada lista e seu respectivo lock
65      for (int i = 0; i < M_LISTS; ++i) {
66          lists[i].head = NULL;
67          omp_init_lock(&locks[i]); // Inicializa o lock
68      }
69
70      #pragma omp parallel for
71      for (int i = 0; i < N_INSERTIONS; ++i) {
72          unsigned int seed = (unsigned int)time(NULL) ^ omp_get_thread_num();
73
74          int value_to_insert = rand_r(&seed) % 1001;
75          int list_index = rand_r(&seed) % M_LISTS; // Escolhe uma das M listas
76
77          // 3. Adquire o lock explícito para a lista escolhida
78          omp_set_lock(&locks[list_index]);
79
80          // --- Início da Região Crítica ---
81          insert(&lists[list_index], value_to_insert);
82          // --- Fim da Região Crítica ---
83
84          // 4. Libera o lock
85          omp_unset_lock(&locks[list_index]);
86      }
87
88      printf("Inserções concluídas.\n");
89
90      long long total_count = 0;
91      for (int i = 0; i < M_LISTS; ++i) {
92          long count = 0;
93          for (Node* current = lists[i].head; current != NULL; current = current-
    >next) count++;
94          total_count += count;
95      }
96      printf("Total de inserções: %lld (esperado: %d)\n", total_count,
    N_INSERTIONS);
97
98      // 5. Destrói os locks e libera toda a memória
99      for (int i = 0; i < M_LISTS; ++i) {
100         omp_destroy_lock(&locks[i]);
```

```
        free_list(&lists[i]);
    }
    free(lists);
    free(locks);

    return 0;
}
```