Duas_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;

// 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) {
        fprintf(stderr, "Falha na alocação de memória\n");
        return;
    }
    newNode->data = value;
    newNode->next = list->head;
    list->head = newNode;
}

// 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;

    // Inicializa as duas listas com a cabeça apontando para NULL
    LinkedList listA = { NULL };
    LinkedList listB = { NULL };

    // A diretiva 'parallel for' distribui as iterações entre as threads
    #pragma omp parallel for
    for (int i = 0; i < N_INSERTIONS; ++i) {
        // Cada thread precisa de sua própria seed para rand_r ser thread-safe
        unsigned int seed = (unsigned int)time(NULL) ^ omp_get_thread_num();
```

```c

        int value_to_insert = rand_r(&seed) % 1001; // Valor aleatório de 0 a 1000
        int list_choice = rand_r(&seed) % 2;       // Escolha aleatória: 0 ou 1

        if (list_choice == 0) {
            // Região Crítica Nomeada para a lista A.
            #pragma omp critical (lock_A)
            {
                insert(&listA, value_to_insert);
            }
        } else {
            // Região Crítica Nomeada para a lista B.
            #pragma omp critical (lock_B)
            {
                insert(&listB, value_to_insert);
            }
        }
    }

    printf("Inserções concluídas.\n");

    // Contagem para verificação
    long countA = 0;
    for (Node* current = listA.head; current != NULL; current = current->next) countA++;
    long countB = 0;
    for (Node* current = listB.head; current != NULL; current = current->next) countB++;

    printf("Elementos na Lista A: %ld\n", countA);
    printf("Elementos na Lista B: %ld\n", countB);
    printf("Total de inserções: %ld (esperado: %d)\n", countA + countB, N_INSERTIONS);

    // Libera a memória alocada
    free_list(&listA);
    free_list(&listB);

    return 0;
}
```