Name : KAMCHE YANN ARNAUD

Matricule: FE21A208

Task: Implement a Random number generator.

Brief Description of what is a random number generator

A random number generator using hash functions is a type of generator that uses a hash function to produce a sequence of seemingly random numbers. Hash functions are typically deterministic functions that map an input of arbitrary size to a fixed-size output. By feeding a hash function with a sequence of input values, we can produce a sequence of output values that appear to be random.

To use a hash function as a random number generator, we typically start with a seed value and then apply the hash function iteratively to produce a sequence of random numbers. The seed value is used as the first input to the hash function, and each subsequent input is generated by applying the hash function to the previous output value.

Some properties of a good random generator are:

- Uniformity: The output values should be uniformly distributed across the output range.

- Independence: The output values should be independent of each other.

- Sensitivity: Small changes in the input should produce large changes in the output.

- Non-reversibility: It should be computationally infeasible to determine the input from the output.

.

Screenshot of Code

In the following code, I implemented the random generator using Queues and pointers in C

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define QUEUE_CAPACITY 10
#define HASH_TABLE_SIZE 256
typedef struct Node {
    int value;
    struct Node* next_node;
} Node;

typedef struct Queue {
    Node* head_node;
    Node* tail_node;
    size_t queue_size;
} Queue;

void enqueue(Queue* queue, int value) {
    Node* new_node = (Node*) malloc(sizeof(Node));
    new_node->value = value;
    new_node->next_node = NULL;
    if (queue->head_node == NULL) {
        queue->head_node = new_node;
        queue->tail_node = new_node;
    } else {
        queue->tail_node->next_node = new_node;
        queue->tail_node = new_node;
    }
    queue->queue_size++;
}
int dequeue(Queue* queue) {
    if (queue->head_node == NULL) {
        fprintf(stderr, "Error: queue is empty\n");
```

```
        exit(EXIT_FAILURE);
    }
    int value = queue->head_node->value;
    Node* temp_node = queue->head_node;
    queue->head_node = queue->head_node->next_node;
    free(temp_node);
    queue->queue_size--;
    return value;
}

void hash_function(int* input_values, size_t input_size, Queue** hash_table) {
    for (size_t i = 0; i < input_size; i++) {
        int index = i % HASH_TABLE_SIZE;
        int value = *(input_values + i);
        Queue* queue = *(hash_table + index);
        if (queue->queue_size == QUEUE_CAPACITY) {
            dequeue(queue);
        }
        enqueue(queue, value);
    }
}
void print_hash_table(Queue** hash_table) {
    for (int i = 0; i < HASH_TABLE_SIZE; i++) {
        Queue* queue = *(hash_table + i);
        if (queue->queue_size > 0) {
            printf("Hash[%d]:", i);
            Node* current_node = queue->head_node;
            while (current_node != NULL) {
                printf(" %d", current_node->value);
                current_node = current_node->next_node;
            }
            printf("\n");
        }
    }
}
```

```
void free_hash_table(Queue** hash_table) {
    for (int i = 0; i < HASH_TABLE_SIZE; i++) {
        Queue* queue = *(hash_table + i);
        Node* current_node = queue->head_node;
        while (current_node != NULL) {
            Node* temp_node = current_node;
            current_node = current_node->next_node;
            free(temp_node);
        }
        free(queue);
    }
}
int main() {
    srand(time(NULL));

    size_t input_size = 10;
    int* input_values = (int*) malloc(input_size * sizeof(int));
    for (size_t i = 0; i < input_size; i++) {
        *(input_values + i) = rand();
    }
    Queue** hash_table = (Queue**) calloc(HASH_TABLE_SIZE, sizeof(Queue*));
    for (int i = 0; i < HASH_TABLE_SIZE; i++) {
        *(hash_table + i) = (Queue*) malloc(sizeof(Queue));
        (*(hash_table + i))->head_node = NULL;
        (*(hash_table + i))->tail_node = NULL;
        (*(hash_table + i))->queue_size = 0;
    }
    hash_function(input_values, input_size, hash_table);
    printf("Input:");
    for (size_t i = 0; i < input_size; i++) {
        printf(" %d", *(input_values + i));
    }
    printf("\n");
    print_hash_table(hash_table);
    free(input_values);
```

```
    free_hash_table(hash_table);
    free(hash_table);
    return 0;
}
```

Results