```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_BLOCK_SIZE 4096
#define MIN_BLOCK_SIZE 512
#define MAX_POINTERS   10000

// Structure to represent each data block
typedef struct {
    char *data;
    size_t size;
} Block;

// Structure to represent each pointer (location and overflow location)
typedef struct {
    char *location;
    char *overflow_location; // NULL if unused
} Pointer;

// Structure to hold all pointers for retrieval
typedef struct {
    Pointer pointers[MAX_POINTERS];
    int count;
} PointerArray;

// Function prototypes
Block *split_file_into_blocks(const char *filepath, int *num_blocks);
```

```
char *generate_random_location(size_t size);
int check_storage_collision(const char *location, size_t size);
int is_location_sufficient(const char *location, size_t size);
void write_block(const char *location, Block block);
void write_overflow(const char *location, Block block);
void store_pointer_key(PointerArray *key, const char *key_storage);
Block read_block(const char *location);
Block merge_blocks(Block part1, Block part2);
void reassemble_file(Block *blocks, int count, const char *output_path);

// Store file into cloud storage, with pointers and block separation
void store_file(const char *filepath, const char *key_storage) {
    int i, num_blocks;
    PointerArray pointer_array = { .count = 0 };
    Block *blocks = split_file_into_blocks(filepath, &num_blocks);

    // For each block, store it securely
    for (i = 0; i < num_blocks; i++) {
        char *loc = NULL;
        do {
            loc = generate_random_location(blocks[i].size);  // Randomize
storage location
        } while (check_storage_collision(loc, blocks[i].size)); // Check for
collisions

        if (is_location_sufficient(loc, blocks[i].size)) {  // If space is
sufficient
            write_block(loc, blocks[i]);  // Store the block
            pointer_array.pointers[pointer_array.count++] = (Pointer){ loc,
NULL };
```

```
        } else {  // Handle overflow (split the block)
            size_t half = blocks[i].size / 2;
            Block part1 = { blocks[i].data, half };
            Block part2 = { blocks[i].data + half, blocks[i].size - half };

            write_block(loc, part1);  // Write first part
            char *overflow_loc = generate_random_location(part2.size);  // Get new overflow location
            write_block(overflow_loc, part2);  // Write overflow part
            pointer_array.pointers[pointer_array.count++] = (Pointer){ loc, overflow_loc };  // Add pointers
        }
    }

    // Store the pointers securely
    store_pointer_key(&pointer_array, key_storage);
    for (int i = 0; i < num_blocks; i++) {
        free(blocks[i].data);
    }
    free(blocks);
}

// Retrieve a file from storage based on pointers
void retrieve_file(PointerArray *key, const char *output_path) {
    Block *blocks = malloc(sizeof(Block) * key->count);

    // Retrieve each block and merge if necessary
    for (int i = 0; i < key->count; i++) {
        Block part1 = read_block(key->pointers[i].location);
```

```
        if (key->pointers[i].overflow_location != NULL) {
            Block part2 = read_block(key->pointers[i].overflow_location);
            blocks[i] = merge_blocks(part1, part2);  // Merge blocks if
overflow exists
        } else {
            blocks[i] = part1;
        }
    }

    // Reassemble and write the file
    reassemble_file(blocks, key->count, output_path);
    for (int i = 0; i < key->count; i++) {
        free(blocks[i].data);
    }
    free(blocks);
}
```