```
#ifndef BLOCKS_WORLD_H
#define BLOCKS_WORLD_H

void create(int N);
int isOnTable(int m);
int isOpen(int m);
int isOn(int m);
int isAbove(int m, int n);
void move(int m, int n);

#endif // BLOCKS_WORLD_H

#include "blocks_world.h"
#include <stdlib.h>

static int* world;
static int size;

void create(int N) {
    world = malloc((N + 1) * sizeof(int));
    size = N;
    for (int i = 1; i <= N; i++) {
        world[i] = 0; // All blocks are initially on the table
    }
}

int isOnTable(int m) {
    if (m <= 0 || m > size) return 0;
    return world[m] == 0;
}

int isOpen(int m) {
    if (m == 0) return 1; // The table is always open
    if (m < 0 || m > size) return 0;
    for (int i = 1; i <= size; i++) {
        if (world[i] == m) return 0;
    }
    return 1;
}

int isOn(int m) {
    if (m <= 0 || m > size) return -1; // Invalid block number
    return world[m];
}
```

```
int isAbove(int m, int n) {
    if (m <= 0 || n <= 0 || m > size || n > size) return 0;
    int current = m;
    while (world[current] != 0) {
        if (world[current] == n) return 1;
        current = world[current];
    }
    return 0;
}

void move(int m, int n) {
    if (m <= 0 || n < 0 || m > size || n > size || !isOpen(m) || !isOpen(n)) return;
    if (n == 0 || world[n] == 0) {
        world[m] = n;
    }
}
```

```
CC=gcc
CFLAGS=-Wall -g
OBJ=blocks_world.o main.o

all: program

blocks_world.o: blocks_world.c blocks_world.h
	$(CC) $(CFLAGS) -c blocks_world.c

main.o: main.c
	$(CC) $(CFLAGS) -c main.c

program: $(OBJ)
	$(CC) $(CFLAGS) -o blocks_world $(OBJ)

clean:
	rm -f *.o blocks_world

.PHONY: all clean program
```

```
#include "blocks_world.h"
#include <stdio.h>

int main() {
    // Create a world with 12 blocks
    create(12);
```

```
    // Create the first tower with blocks 1, 2, and 3
    move(2, 1); // Move block 2 onto block 1
    move(3, 2); // Move block 3 onto block 2

    // Create the second tower with blocks 4, 5, 6, and 7
    move(5, 4); // Move block 5 onto block 4
    move(6, 5); // Move block 6 onto block 5
    move(7, 6); // Move block 7 onto block 6

    // Create the third tower with blocks 8, 9, 10, 11, and 12
    move(9, 8);  // Move block 9 onto block 8
    move(10, 9); // Move block 10 onto block 9
    move(11, 10); // Move block 11 onto block 10
    move(12, 11); // Move block 12 onto block 11

    // Print the status of each block
    for (int i = 1; i <= 12; i++) {
        printf("Block %d is on block %d\n", i, isOn(i));
    }

    // Optionally, check if specific isAbove conditions are met
    if (isAbove(3, 1)) {
        printf("Block 3 is correctly placed above Block 1\n");
    }
    if (isAbove(7, 4)) {
        printf("Block 7 is correctly placed above Block 4\n");
    }
    if (isAbove(12, 8)) {
        printf("Block 12 is correctly placed above Block 8\n");
    }

    return 0;
}

#include <criterion/criterion.h>
#include "blocks_world.h"

Test(blocks_world, create_and_initial_conditions) {
    create(10);
    for (int i = 1; i <= 10; i++) {
        cr_assert(isOnTable(i), "Block %d should be on the table initially.", i);
        cr_assert(isOpen(i), "Block %d should be open initially.", i);
    }
```

```
}

Test(blocks_world, isOnTable) {
    create(5);
    // Check that all blocks are on the table initially
    for (int i = 1; i <= 5; i++) {
        cr_assert(isOnTable(i), "Block %d should be on the table after creation.", i);
    }
    // Edge cases
    cr_assert_not(isOnTable(0), "isOnTable(0) should return false.");
    cr_assert_not(isOnTable(6), "isOnTable(6) should return false for out-of-range block.");
}

Test(blocks_world, isOpen) {
    create(4);
    // Initially, all should be open
    for (int i = 1; i <= 4; i++) {
        cr_assert(isOpen(i), "Block %d should be open initially.", i);
    }
    // Move block 3 onto block 2
    move(3, 2);
    cr_assert_not(isOpen(2), "Block 2 should not be open after block 3 is moved onto it.");
    cr_assert(isOpen(3), "Block 3 should still be open.");
    cr_assert(isOpen(0), "Table should always be open.");
    // Edge case for out-of-range block
    cr_assert_not(isOpen(5), "isOpen(5) should return false for out-of-range block.");
}

Test(blocks_world, isOn) {
    create(6);
    move(4, 2); // Place block 4 on block 2
    move(2, 1); // Place block 2 on block 1, making a tower 1-2-4
    cr_assert_eq(isOn(4), 2, "Block 4 should be on block 2.");
    cr_assert_eq(isOn(2), 1, "Block 2 should be on block 1.");
    cr_assert_eq(isOn(1), 0, "Block 1 should be on the table (0).");
    // Edge cases
    cr_assert_eq(isOn(0), -1, "isOn(0) should return -1 (invalid).");
    cr_assert_eq(isOn(7), -1, "isOn(7) should return -1 for out-of-range block.");
}

Test(blocks_world, isAbove) {
    create(5);
    move(2, 1); // Block 2 on Block 1
    move(3, 2); // Block 3 on Block 2, creating a stack 1-2-3
```

```
    cr_assert(isAbove(3, 1), "Block 3 should be above block 1.");
    cr_assert(isAbove(2, 1), "Block 2 should be above block 1.");
    cr_assert_not(isAbove(1, 3), "Block 1 is not above block 3.");
    cr_assert_not(isAbove(5, 1), "Block 5 is not above block 1 and should return false.");
    // Edge cases for non-existing blocks
    cr_assert_not(isAbove(6, 1), "isAbove(6, 1) should return false for out-of-range block.");
    cr_assert_not(isAbove(1, 6), "isAbove(1, 6) should return false for out-of-range block.");
}

Test(blocks_world, move) {
    create(3);
    move(1, 0); // Should have no effect since it's already on the table
    move(2, 3); // Valid move: Block 2 onto Block 3
    cr_assert_eq(isOn(2), 3, "Block 2 should now be on block 3.");
    move(3, 2); // Invalid move, as block 2 is not open
    cr_assert_eq(isOn(3), 0, "Block 3 should still be on the table (0).");
    // Test moving onto non-open block
    move(1, 2); // Block 1 cannot move onto block 2 as block 2 is not open
    cr_assert_eq(isOn(1), 0, "Block 1 should remain on the table.");
    // Edge case
    move(4, 1); // Non-existing block move should have no effect
    cr_assert_eq(isOn(4), -1, "Moving non-existent block should not change anything.");
}
```