```
// Assignment #8
// This program allows us to have our robots give the colony ice and info

// Include library
#include <stdio.h>
#include <stdlib.h>

// Structure for robots, map, and location
struct robot {
   int range;
   int capacity;
};
struct map {
   int range1;
   int ice;
};
struct location {
   int i;
   int j;
};

// max and min functions
int max (int x, int y)
{
   if (x > y) return x;
   return y;
}
int min (int x, int y)
{
   if (x < y) return x;
   return y;
}

// Variable declaration
struct robot rcall[4];
struct map dest[3][3];
int total_ice;
int ice_collected;

// map status() function:
// Displays current status of map
void map_status() {
   // Variable declaration for map
```

```
    int i = 0, j = 0;
    printf("\n");

    // Loop through map location array
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            if (dest[i][j].range1 > 0)
                printf("%du\t", dest[i][j].range1);
            else if (dest[i][j].ice > 0)
                printf("%di\t", dest[i][j].ice);
            else
                printf("-\t");
        }
        printf("\n");
    }
}
// robot status():
// Update map with robot status
void robot_status() {
    int i = 0;
    printf("\n");

    // Header of map layout
    printf("Robot\tRange\tCapacity\n");

    // display values of robot
    for (i = 0; i < 4; i++){
        printf("%3d\t%2d\t%3d\n",(i + 1), rcall[i].range, rcall[i].capacity);
    }
}
// populate map robot() function:
// reads the data from the file and
// then populate the array structures robotnum, location;
void populate_map_robot() {
    char filename[50];
    int i = 0, j = 0;
    FILE *ifp = NULL;

    // while loop to read file, if it does not exist clear window
    while (ifp == NULL){
        printf("What is the name of the map file?\n");
        scanf("%s", &filename);
        ifp = fopen(filename, "r");
        if (!ifp){
            system("cls");
        }
    }
        // Loop to read map
        for (i = 0; i < 3; i++) {
            for (j = 0; j < 3; j++) {
                fscanf(ifp,"%d%d", &dest[i][j].range1, &dest[i][j].ice);
                total_ice += dest[i][j].ice;
            }
        }
        // Loop to read the four robots
        for (i = 0; i < 4; i++) {
            fscanf(ifp, "%d%d", &rcall[i].range, &rcall[i].capacity);
        }
    fclose(ifp);
}
// enter_location() function: input a number returns the structure location
struct location enterlocation(int robotno) {
    // Structure variable for location
    struct location robot_loc;

    // Ask user for next robot
    printf("\nWhere would you like to send robot#%d?", robotno);
    map_status();
    printf("\n");

    // Read the locations
    scanf("%d %d", &robot_loc.i, &robot_loc.j);
    // fitting coordinate to 0 to n-1 board
    robot_loc.i -= 1;
    robot_loc.j -= 1;

    return robot_loc;
};
// Location_of_ice(): takes in a structure location and
// an integer variable robot number
// This function tell us the if ice was located and how much to get to
// capacity or do nothing
void location_of_ice(struct location r_loc, int robotno) {
    // Check depth range of ice at location (i,j) is greater than the surface

    if (dest[r_loc.i][r_loc.j].range1 > 0) {
```

```c
        // What else can be known about the depth
        // if (-) we cant dig if (+) we can
        int depth = max(0, dest[r_loc.i][r_loc.j].range1 - rcall[robotno].range);
        // Update map location
        dest[r_loc.i][r_loc.j].range1 = depth;

        // Show info
        printf("\nThe robot has scanned this section and located %s ice!\n",
            (depth != 0) ? "some" : "no");
    }
    // Check if the ice buried is greater than zero at its location
    else if (dest[r_loc.i][r_loc.j].ice > 0) {

          // Get the remaining info for amount of ice
          int robcap = min(rcall[robotno].capacity, dest[r_loc.i][r_loc.j].ice);

          // Calculate ice collected
          total_ice += robcap;

          // Tell user if the robot is taking some or all the ice back
          printf("\nThe robot takes %s of the ice back to the storage facility!\n",
              robcap != dest[r_loc.i][r_loc.j].ice ? "some" : "all");

          dest[r_loc.i][r_loc.j].ice = max(0, dest[r_loc.i][r_loc.j].ice - rcall[robotno].capacity);
    }
    // if not, tell user it has been cleared
   else {
      printf("\nThis section has already been cleared.\n");
   }
}
// no more ice() function: checks through a boolean expression
// 0: false 1: true, the function checks if all the ice is
// collected
int no_more_ice(struct location r_loc) {
    // declare index variables
    int i = 0, j = 0;

    // Loop through locations
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++){
            // if the ice at a certain location is not empty return zero
            if (dest[i][j].ice > 0) {
                return 0;
            }
        }
    }
    // Otherwise return 1
    return 1;
}
// Main function
int main(void)
{
    // 9 locations of ice.
   // int count = 9;
   //we ended up using total Ice instead of my count variable to keep track
   //if how many time we loop.
    int flag = 0;
    int i = 0;
    populate_map_robot();

    while(total_ice != 0) {
        robot_status();

        for (int i = 1; i <= 4; i++) {
            struct location coordinate;
            coordinate = enterlocation(i);
            location_of_ice(coordinate, i);
            if (no_more_ice(coordinate) == 1) {
                flag = 1;
                break;
        }
    }
    if (flag == 1) {
        break;
        total_ice--;
    }
    }
    if (total_ice = ice_collected) {
        printf("All of the ice in the surrounding area has been moved to your colony storage
facility!");
    }
    return 0;
}
```