Assignment #8

Introduction to C Programming – COP 3223

Objectives

1. To learn how to design and implement functions for program development
2. To reinforce how to use pass by value and pass by reference variables
3. To learn how to use structures to store information and solve problems

Introduction: Mission to Mars

Your friend has been playing a new Mars Colony simulator nonstop! They are always talking about how cool it would be if they could be a on the first real-life mission to Mars! To amuse your friend, you have decided to create a series of programs about the possible first colony on Mars.

Problem: Stock up on Water (marswater.c)

Congratulations! Our Mars colony is established and ready for human residents. Before they get here we should stock up on as much ice as we can. Ice can be melted and purified for our human residents to use for drinking, cooking, and cleaning.

Our autonomous robots can search for and detect deposits of ice on the surface of the red planet. However, they can only detect ice within a fairly narrow range. Likewise, they can only carry a certain amount of ice back to the storage facilities.

In this program we want to send our robots to check the areas around our colony and return with any ice and information that they find.

Program Details

The colony has four robots that detect and retrieve ice deposits. Each robot has a specific range that it can search in and a certain amount of ice that they can carry. Some ice may be too buried for a robot to detect and some ice deposit may be too large for a robot to retrieve.

Your program will searching the surrounding area for ice deposit and managing the robots to retrieve them. The colony's accessible area has been split into a 3x3 grid of spaces to search. Not all spaces are guaranteed to have ice in them. The user will simulate the management program to track how deep each deposit of ice is buried. For each hour, the user will need to choose which section of the area to send each available robot to. As each robot is deployed, your program should update the map.

In order to determine if ice is available, we need to send a robot with a sufficiently high detection range to that space. Then, the ice can be picked up by any robot. Ice may be picked up in pieces, so even if the robot does not have enough capacity to carry all of it, they can move some of the ice to the storage facility.

The program should run until all of the ice has been picked up for the colony.

Consider the following example:

4 Range **2 Ice**	1 Range 1 Ice	3 Range 1 Ice
1 Range 1 Ice	2 Range 1 Ice	3 Range 1 Ice
1 Range 1 Ice	5 Range 3 Ice	1 Range 1 Ice

There are 2 cubic feet of ice, buried 4 feet deep in the upper left corner.

Our autonomous robots might have the following qualities:

0	1	2	3
Range: 3 Capacity: 1	Range:2 Capacity: 5	Range: 5 Capacity: 1	Range: 1 Capacity: 3

Robot 2 is the only robot that can detect that there is ice in the upper left corner of out map. However, it can only carry one cubic feet (one unit) of ice. In the next hour, the management program should send a different robot to that square. Either robot 1 or robot 3 would be ideal, because they have enough capacity to retrieve both units of ice.

Your program should prompt the user for instructions for each crew robot. Ask which section of the map they are being sent to. Let them know if there any ice has been detected in that section yet. If no ice has been detected, the robot must use their detect action. Once ice has been detected, the ice can be picked up. A robot can only transport an amount of ice equal to their capacity value.

Your program should run until all the ice has been picked up.

Implementation Restrictions

You must use the following structures to store information:

```
struct robot
{
      int range;
      int capacity;
};

struct map
{
      int range;
      int ice;
};
```

It is not sufficient to simply have the structures in your program, you must use them store information and process operations for the program. You may use type definition if you prefer.

Though function prototypes will not be provided, it is expected that you follow good programming design and create **at least four functions** with well-specified tasks related to the solution of this problem. Make sure to pay very careful attention to parameter passing.

Hints

Create a 3x3 array for the map. The value range indicates how deep any ice in that section is buried. A robot who can sense to that depth is required to verify whether there is any ice in that section or not. The value ice indicates how much ice is in that location. Both values will be provided by an input file. See below for specifications.

Create a 1x4 array for your robots. These values will also be provided by an input file. Consider making an initialization function to process this file and initialize your data structures for the program.

When a section has been scanned, reduce the range value of that section to 0 so that you can tell it has been scanned already.

Input Specification

The input file will contain 9 pairs of integers representing the range and ice values (in that order) for each square. This will be followed by 4 pairs of integers representing the range and capacity values (in that order) for the robots. Here is the sample file that matches the above example:

```
4 2     1 1     3 1
1 1     2 1     3 1
1 1     5 3     1 1
3 1
2 5
5 1
1 3
```

Remember to prompt the user for the name of the input file and open the file that they specify.

The user will input directions in a 1-based, left to right fashion:

```
1 1     1 2     1 3
2 1     2 2     2 3
3 1     3 2     3 3
```

Output Specification

At the start of each hour, display the status of the four robots:

```
Robot   Range   Capacity
1       X1      Y1
2       X2      Y2
3       X3      Y3
4       X4      Y4
```

Then, for each robot, show the user the current state of the map:

```
4u      1u      3u
1u      2u      3u
1u      5u      1u
```

(example initial map; use u to indicate an area is UNSCANNED)

2I 1u 3u
1u 1I 1I
1u 5u 1u

(user has scanned sections 1 1, 2 2, and 2 3; use I to indicate ice has been detected)

- 1s 3s
1s 1T 1I
1s 5s 1s

(player has both scanned and retrieved the ice from section 1 1; use - to indicate no actions remain for that section)

Then prompt the user with a robot number and ask where they should be sent:

> Where would you like to send robot X?

Immediately resolve this by sending the robot to the corresponding section on the map. If the section has not been scanned yet, check the range of the robot. If their range is not equal to or greater than the range required for that space, there is nothing the robot can do. If the range is sufficient, mark the section as scanned. If the section has already been scanned in a previous hour or by a previous robot, the current robot can pick up the ice. They can only carry as much ice as their capacity allows.

> The robot has not found any ice.
>
> The robot has scanned this section and located some ice!
>
> The robot takes all of the ice back to the storage facility!
>
> The robot takes some of the ice back to the storage facility.
>
> This section has already been cleared.

When all of the ice has been retrieved, print:

> All of the ice in the surrounding area has been moved to your colony storage facility!

Input/Output Sample(s)
Sample input and output files will be provided on the webcourse.

Acceptable Resources
Remember, the use of online help sites is strictly prohibited. The only acceptable resources for these assignments are below:

- Course Webcourse
    - In particular: Week 13 – Strings and Structures: Concepts, and Weeks 14/15 – Strings and Structures: Examples

- Course Textbook
    - Programming Knights: An Introduction to Programming in Python and C by Arup Guha
- Professor Guha's Course Archive
    - http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3223/
- Course TAs and Instructor Office Hours
    - Getting Help: https://webcourses.ucf.edu/courses/1336411/pages/getting-help

Style Notes

Please review the course Style Guide on the webcourse, with special attention to the following notes:

- remove excess comments and instructions, leaving pre- and post-conditions and comments in the main function. Add comments to the supplemental functions you write as normal.
- comment major sections of code addressing: "What does this block do?" and "Why did I implement this block in this way?"
- place comments above the line(s) to which it applies
- use inline comments (//) and leave one space between // and the comment's first character
- All variables should be declared at the top of your functions and should have meaningful names
- Indent the contents of functions four spaces or one tab
- leave a space on both sides of any binary or conditional operators you use in your code (i.e., operators that take two operands).
- keywords if, while, and for should have a single space after them
- contents of if statements and loops should be indented four spaces or one tab
- conditions should not have any space immediately after each `(` or immediately before each `)`.

Deliverables

One source file: *marswater.c* for your solution to the given problem submitted over WebCourses.

Restrictions

Although you may use other compilers, your program must compile and run using Code::Blocks. Your program should include a header comment with the following information: your name, course number, section number, and assignment title. Also, make sure you include comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

3) Compatibility – You must submit C source files that can be compiled and executed in a standard C Development Environment. If your program does not compile, you will get a sizable deduction from your grade.