1775396 🡨 student ID

Ryan Peters  
Assignment 2: Recursion

Contents

[Design 1](#_Toc497942622)

[Show and explain output 2](#_Toc497942623)

[Compiling and running instructions 3](#_Toc497942624)

## Design

My approach to this design was probably against the spirit of this assignment. By this I mean I had to manually identify the conditions under which the recursion could become trapped in a loop.

I initially found that the conventional way of thinking about this problem is to represent the number of explorers, cannibals, and boat that are present on the left side of the river as a vector of three integers. For example, we can express the starting condition of the problem as:

<numExplor, numCanni, binaryBoat> or <3,3,1>

In order to track the changes to my left side state vector, which I named curState, I set my recursive call to pass a vector<vector<int>> object as the return type, where each recursive call would add its own updated and validated state to the vector<vector<int>> and pass it on back.

Also note, that the third index of the vector is treated like a binary bit, so it will always be either 0 or 1. As a result, when it is 0, it is treated as if the boat is on the opposite shore and any modifications to the left hand state will become additions, as opposed to the subtractions you would get when the boat is on the left shore.

As a result, the base case will be when the left shore state is <0, 0, 0>.

The recursive conditions get rather messy however, as I found that there are 5 possible ways in which any one state can be modified before checking if the results are valid. It will be easier from here on to show in pseudo code than to describe conventionally.

Set initial state to be <3,3,1>, and name if curState;

Set a global const “vector<vector<int>> options” object to be {{1,1,1},{1,0,1},{0,1,1},{2,0,1},{0,2,1}};

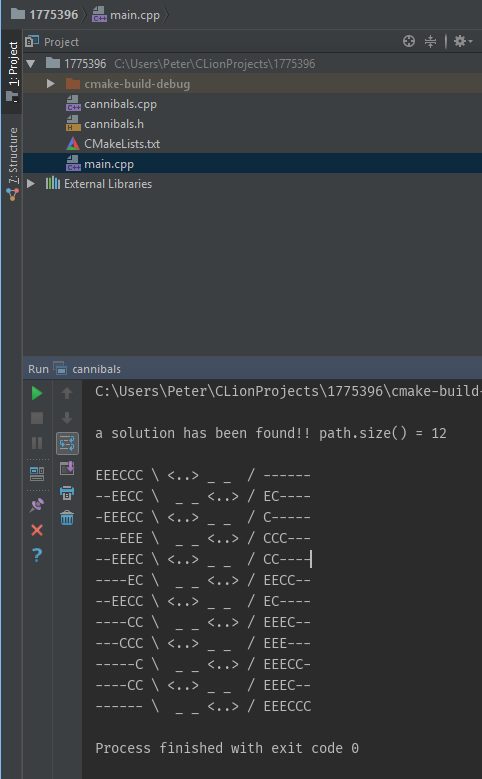
Begin recursively modifying the curState, where at each recursive call we take the given curState and create a local member as a subset of the global options member. This subset will consist only of the options that don’t result in an immediate invalid state.

Using a for-each loop, make recursive calls to our recursiveSolution function passing our updated curState object and one of our prevalidated modification options.

Presto-chango we get a solution!

## Show and explain output

Figure 1: command line output of cannibals program



Here’s a rough looking copy/past from the command line of the output:

EEECCC \ <..> \_ \_ / ------ Each ‘E’ character represents an explorer

--EECC \ \_ \_ <..> / EC---- Each ‘C’ character represents a cannibal

-EEECC \ <..> \_ \_ / C----- The “<..>” represents the boat with no occupants

---EEE \ \_ \_ <..> / CCC---

--EEEC \ <..> \_ \_ / CC---- The boat is never shown with any occupants as I found

----EC \ \_ \_ <..> / EECC-- that showing occupants in the transit stage made the

--EECC \ <..> \_ \_ / EC---- output harder for the eye to read.

----CC \ \_ \_ <..> / EEEC--

---CCC \ <..> \_ \_ / EEE--- The ‘\_’ character portrays water, used as a space filler to

-----C \ \_ \_ <..> / EEECC- help keep the output consistent in its spacing.

----CC \ <..> \_ \_ / EEEC--

------ \ \_ \_ <..> / EEECCC The ‘-‘ character portrays dirt, serves the same purpose as ‘\_’

## Compiling and running instructions

In order to compile the source files from the command line:

g++ -o RCPeters\_1775396 -std=c++17 \*.cpp \*.h

🡪This command assumes that there are no .cpp or .h files in the directory other than the source files contained in 1775396.zip

To now run the executable from the command line, enter the following command.

./RCPeters\_1775396

You should get an output that matches that in Figure 1 above.