Assignment 2

By

Robert Szkutak

The Problem:

A group of three ethical computer hackers are being sued by a group of three lawyers representing a large company over a misunderstanding of potential copyright infringement. All six people are at the hotel and need to be transported to the courtroom. A car can transport them, but it can only take two people at a time and it must always have at least one person in it driving the car between the courtroom and the hotel. In order to win their case and clear their names the hackers must stick together. They must never allow the lawyers to outnumber them in any location at any time. The hackers must find a way to get themselves and the lawyers transported to the courtroom while abiding under this condition.

An Imperative Solution in C++:

```
#include <iostream>

#define MAXPEOPLE 6

using namespace std;

class Person
{

      public:
            Person();
            ~Person();
            int id;//1 == Hackers , 2 == Lawyers
            int place;//1 == Hotel , 2 == Courtroom

};

Person::Person() {}
Person::~Person() {}

bool success(Person People[]);
void moveRight(Person People[], int hackers, int lawyers);
void moveLeft(Person People[], int hackers, int lawyers);
void printStatus(Person People[]);

int main()
{
     Person People[MAXPEOPLE];
```

```
for (int i = 0; i < MAXPEOPLE; i++)
{
   People[i].place = 1;

   if(i < MAXPEOPLE / 2)
       People[i].id = 1;
   else
       People[i].id = 2;

}

printStatus(People);
moveRight(People, 1, 1);
printStatus(People);
moveLeft(People, 1, 0);
printStatus(People);
moveRight(People, 0, 2);
printStatus(People);
moveLeft(People, 0, 1);
printStatus(People);
moveRight(People, 2, 0);
printStatus(People);
moveLeft(People, 1, 1);
printStatus(People);
moveRight(People, 2, 0);
printStatus(People);
moveLeft(People, 0, 1);
printStatus(People);
moveRight(People, 0, 2);
printStatus(People);
moveLeft(People, 0, 1);
printStatus(People);
```

```
    moveRight(People, 0, 2);
    printStatus(People);

    if(!success(People))
        cout << "\n\nERROR!!!";

    cin >> People[0].id;

    return 0;

}

bool success(Person People[]){

    bool ret = true;

    for(int i = 0; i < MAXPEOPLE; i++)
    {
        if(People[i].place == 1)
            ret = false;
    }

    return ret;

}

void moveRight(Person People[], int hackers, int lawyers){

    for(int i = 0; i < MAXPEOPLE; i++)
    {
        if(People[i].id == 1 && People[i].place == 1 && hackers > 0)
        {
            People[i].place = 2;
            hackers--;

        }


        if(People[i].id == 2 && People[i].place == 1 && lawyers > 0)

        {

            People[i].place = 2;

            lawyers--;

        }

    }

}


void moveLeft(Person People[], int hackers, int lawyers){


    for(int i = 0; i < MAXPEOPLE; i++)

    {

        if(People[i].id == 1 && People[i].place == 2 && hackers > 0)

        {

            People[i].place = 1;

            hackers--;

        }


        if(People[i].id == 2 && People[i].place == 2 && lawyers > 0)

        {

            People[i].place = 1;

            lawyers--;

        }

    }

}


void printStatus(Person People[])

{

    int hotelhcount = 0, courthcount = 0;

    int hotellcount = 0, courtlcount = 0;
    for(int i = 0; i < MAXPEOPLE; i++)
    {
        if(People[i].id == 1 && People[i].place == 1)
            hotelhcount++;
        if(People[i].id == 1 && People[i].place == 2)
            courthcount++;
        if(People[i].id == 2 && People[i].place == 1)
            hotellcount++;
        if(People[i].id == 2 && People[i].place == 2)
            courtlcount++;
    }

    cout << "Hotel: " << hotelhcount << "H/" << hotellcount << "L";
    cout << "     Court: " << courthcount << "H/" << courtlcount << "L\n\n";

    return;
}
```

A Functional Solution in Lisp:

```
(defun hackersAndLawyers (step)

                              (cond ((equal step 0)
                                 (print "Sending over one hacker and one
lawyer to courtroom")
                              ))

                              (cond ((equal step 1)
                                 (print "Sending one hacker back to hotel")
                              ))

                              (cond ((equal step 2)
                                 (print "Sending over two lawyers to
courtroom")
                              ))

                              (cond ((equal step 3)
                                 (print "Sending back one lawyer to hotel")
                              ))

                              (cond ((equal step 4)
                                 (print "Sending over two hackers to
courtroom")
                              ))

                              (cond ((equal step 5)
                                 (print "Sending back one hacker and one
lawyer to hotel")
                              ))

                              (cond ((equal step 6)
                                 (print "Sending over two hackers to
courtroom")
                              ))

                              (cond ((equal step 7)
                                 (print "Sending back one lawyer to hotel")
                              ))

                              (cond ((equal step 8)
                                 (print "Sending over two lawyers to
courtroom")
                              ))

                              (cond ((equal step 9)
                                 (print "Sending back one lawyer to hotel")
                              ))

                              (cond ((equal step 10)
                                 (print "Sending over two lawyers to
courtroom")
                              ))

                              (incf step)

                              (cond ((< step  11)
                                 (hackersAndLawyers step)
                              ))

)
```

An Analysis of Creating the Two Solutions:

Deriving a solution to the hackers and lawyers problem in both an iterative and functional programming language began similarly but then resulted in taking two very different approaches. The very first thing I did to work towards solving the problem was to make sure I understood the problem myself. I sat down with a whiteboard and a marker and I came up with a step-by-step solution to the problem. I tried to search for distinct patterns in the solution but I was mostly unsuccessful at finding any. By doing all of this, I now felt confident that I completely understood the problem and precisely how it would be solved.

For programming in an iterative language I tried to define the problem in terms of steps needed to be taken to reach the solution. My code ultimately ended up looking exactly like the steps I had come up with on my whiteboard. I simply used my coding instructions to tell the computer step-by-step what to do to solve the problem. I did all the decision making for the computer. The computer simply executed my exact instructions and in the precise order which I specified them in.

For programming the solution in a functional programming language I tried to define the problem in terms of being a function. I reasoned with myself that transporting everyone from one place to another was a single function. There were multiple steps to the process so I decided to take advantage of recursion by incrementing a variable each time through to keep track of what step should occur next. This differed from iterative programming in that even though I wrote the steps to solve the problem the computer was thinking for itself by determining what order to perform the steps to the solution in.

There were some similarities between programming in the two different language paradigms. Both programs ultimately used the same steps in the same order to arrive at the solution to the problem. The key difference was in how that order to perform those steps to the solution was determined; either by myself giving that order to the computer or the computer determining that order on its own.

What I learned from this assignment is that there truly is no best way to solve a problem with a programming language. There are an infinite number of approaches and styles which can be used. I personally felt that in this case the functional programming solution ended up being cleaner and an overall better solution even though it took a bit more brain power on my part to come up with. However, I am sure that this wouldn’t hold true in other cases. As a programmer, it’s my responsibility to understand as many paradigms as I can so that I can solve problems as efficiently and effectively as I can.