Luis Garcia
David Woodward
CSCE 2110.002
10/5/2018

Design Document
Project 1

Design Process

For the design process our group first decided to read the instructions for the project thoroughly and let ideas come to us until we felt comfortable with requirements and created game plan to act on. With our background from 2100; we knew we were going to have to parse through the file, use Boolean bit vectors along with structs to hold personal information for each nurse, and create functions for each set operation. We chose to parse through the file because that is the best and only way we really know how to read-in the files. After some thought, we decided vectors would be the best way to store the shifts for each nurse. We came up with the idea of having a 3D bit vector with the first layer being the 5 weekdays, the second layer being each nurse, and the third layer being every hour of that day. We also decided to have a struct where we would hold the names, initials, and shifts in string for to call upon later. This is what our general design was at the beginning for our group and with that we felt comfortable to start coding and adjust our plans from there based on what would work and what didn't.

Data Structures

The data structures used in this project consisted of structs, 3D bit vectors, 2D int vectors using only 0's and 1's, and 2D string vectors. We created a struct called fullName and inside that struct we saved both the first and last name in a string vector called name, each of their shifts in a string vector called shifts, a string variable called allShifts which combined all the elements from all their shifts into one string, and a string called initials which would hold the initials of the nurse. We created the single string allShifts in order for it to be easier to remove nurses that happened to have the exact same schedule as another nurse.

```
struct fullName {
  vector<string> name;
  vector<string> shifts;
  string allShifts;
  string initials;
};
```

Figure 1: fullName struct

Our 3D boolean vector was called weekDays and we had the most outer layer be set to a size of 5 because we are only working with 5 days with element 0 being monday, element 1 being tuesday and so on. The second

layer was the nurse with 0 being the first nurse read in. As we parsed, every time we got to a new nurse we added them to everyday so the size of the second vector would always be the same. In the third layer of this vector, the shift data was stored. The size will always be 24 for the hours in the day. Depending on what we parsed from the input file, the hours each nurse worked was populated into the third vector layer with 'true' representing they are scheduled to work at that time. So if you wanted to check and see whether a nurse worked on a specific day at a specific time, this would be the format to use weekDays[d][n][h]. For example if weekDays[1][2][10] returned true that means on tuesday our third registered nurse worked during the 10 am hour. Multiple 2D int vectors using 1's and 0's were used for our set operations and check for complete simple coverage. We no longer needed for this vector to be three dimensional because we were just comparing days and hours worked not dealing with the multitude of nurses anymore. Figuring out the minimal schedules seemed a lot easier to do if we could immediately delete any nurses with exact copies of other nurses schedules and the same for shifts. That is why we created the 2D string vectors so we could compare nurses whole shift schedule or individual day shifts by their characters in the string rather than checking through every element of a nurses schedule in our master 3D vector weekDays.

Functionality

1) Parsing

The parsing function reads in the file one line at a time. We wrote the code so that the first line we get we saves the two parts of the nurses name into a vector that will hold the nurses first and last name. For the remaining lines until we get to the empty line, we save that entire string into a vector so that we can later compare nurses schedules easier. Before we move on to the next line we parse the day of the week and the following two numbers that would be the hours the nurses work their shift. This part was a little tricky because we had to change the delimiter used to stop the parser within the same line. Something new we haven't done before but we figured it out. Using the parsed day of the week, we determined what slot in our vector to input values to using a simple if statement. Once we knew where to put our values, we used a for loop to add true values to the hours the nurse worked in our weekDays master vector. We knew where to start and begin from the parsed hour values we got. We got those values in string form so we had to convert them to integers using a c++11 function called stof().

2) Set Functions

Once all of the individual nurse schedules have been read into the master full week schedule (3d bit vector called weekDays), the program can start performing set operations on the data to create the required output data sets. Four set operation functions were created for this project; OR, AND, NOT, and symmetric difference. The following chart shows which set functions were used to create each output data set.

Operation	Name in program	Output Set Created

OR	set_or()	Simple complete coverage (scc)
AND	set_and()	More than 1 nurse (mt1)
NOT	set_not()	No nurses (nn)
Symmetric Difference (XOR)	set_symdiff()	Exactly 1 nurse (e1n)

Table 1: Set Functions

Each function takes in two bit vectors that are passed by reference. The bit vectors are passed by reference here so that the values can be altered within the scope of main. The following image shows an example of the set_or() function. This function works by stepping through every position of the 3d weekDays vector and when the program finds a true value, it sets scc at the corresponding [d][h] position to 1 meaning this hour is covered by at least one nurse. The other three set functions work in similar ways.

```
//set operations
void set_or(vector<vector<vector<bool> > > &weekDays, vector<vector<int> > &scc) {
  for (int d = 0; d < 5; d++) {
    for (int n = 0; n < weekDays[d].size(); n++) {
      for (int h = 0; h < 24; h++) {
        if (weekDays[d][n][h] == true) {
          scc[d][h] = 1;
        }
      }
    }
  }
  return;
}
```

Figure 2: set_or() function

3) Check Simple Complete Coverage

To check for simple complete coverage, we wrote a checkScc() function. This function steps through every position (day and hour) of the simple complete coverage (scc) 2d bit vector created earlier from the set_or() function. If at any point the function finds a 0, it returns false meaning there is a point where no nurse is scheduled. If the function steps through the entire 2d vector and does not find any 0s, the function returns true meaning simple complete coverage is true.

```
bool checkScc(vector<vector<int> > &scc) {
  for (int i=0; i < scc.size(); i++) {
    for (int j=0; j < scc[i].size(); j++) {
      if (scc[i][j] == 0) {
        return false;
      }
    }
  }
  return true;
}
```

Figure 3: checkScc() function

4) Create Minimal Nurse and Shifts Schedule

The algorithm to create the minimal nurse and shift schedule contains 5 main components: create duplicate temporary vectors, delete duplicate nurses from the master schedule, remove each nurse one at a time and test for simple complete coverage (scc), if scc fails without this nurse -> nurse is unique -> add nurse to the minimal nurse schedule. The duplicate temporary vectors are created so that the the program can remove nurses without alerting the master schedule vector. Each loop, the temporary vectors are reset to equal the original master schedule. The next step is to remove any nurses that have the exact schedule as another nurse. This is needed so the next step can function properly. With any duplicate nurses removed, the algorithm now runs through a for loop checking if each nurse is unique. The algorithm removes nurse n from the schedule and tests for simple complete coverage (scc). If scc returns false, we know that the nurse is unique and nurse n is added to the minimal nurse schedule. The final step is to add nurse n back into the schedule so the other nurses can be checked. This is done by resetting the temporary test vectors. The process is the exact same for creating the minimal shift schedule the program just removes one shift at a time instead of a whole nurse.

5) Output

Outputting the data was pretty simple. We wrote a function that takes in a 2d vector and runs a double nested for loop to print all the data/labels in a tabular format. For the table of strings, we hardcoded the column widths, but if we had more time we would have wrote a function that finds the longest string in each column and uses that length to set the column widths.

Work Share

For the work share we met up several times throughout the two weeks we had either at the computer lab or at each other's apartments. The first couple times we met, we discussed our ideas towards the project and the design process came about and the decisions of what data structures to use. To collaborate, we used the text editor Atom's new feature Teletype which allows multiple users to edit a single script in real time much like a

google doc. We decide to have Luis start on the parsing function which wouldn't take too long. The main difference from previous projects was figuring out how we were going to go about parsing a line with string characters and integer values in the same line. David focused on setting up the set operations, and creating the output function to start testing functionality. After Luis finished the parsing, he would join David on working on how to find the minimal schedule of both nurses and shifts. Both of us worked on everything else with the code and even while we were doing our own parts we were right next to each other and able to help one another when questions arose. After we were finally able to complete the code portion of the project, both of us worked on the design document adding our thoughts on all four sections into a understandable cohesive manner.