Project 3

Counting Objects in an Image

By: Tuyet Pham

Design process

Personally, I have more of an external coding style. The only way I can understand a new problem fully is by visually and physically see it right in front of me. This means that I will have to code as I think of solutions. Using a paper and pen method usually help as well. The main idea is that visual cues would aid me in the thinking process.

Although challenging, this project proved to be no different than the ones before. Immediately I started off the project by seeing what I can salvage and reuse from the passed project. No use in trying to reinvent the wheel, when the ones you have is working perfectly fine. Fortunately, I've always been the one to break my programs up into smaller functions, so finding reusable ones didn't break a sweat. After gathering all the possible reusable bits of code, I begin to think of ways to integrate the new problems with old codes.

From reusing the older functions, it decreased the amount of function that were needed to be made down to about 2 (change and display). I knew that I wanted a separate function to **change** and a separate function to **display** the whole chart after those changes were made.

Once I started coding I realized that some of the code for checking the individual pixel within the change function would not be reusable; or at least in ways that would matter. So that piece of code was to be made from scratch.

I will list some more issues below that I have encounter and what I did to fix them –

- I didn't want to complicate things with 1's being in the graph. I solved this by –
    - I made 1's into x's
- My chart was made of strings so when making the object count and having it be able to increment was a bit challenging. I wanted to avoid the whole to_string or atoi by -
    - Making an array of strings that has numbers in it.
    - I had a loop make about 100 strings of digits from 1 to 100.
    - I used a string pointer variable to iterate through the array by the incrementation operator (++) after each object number assignment. This will cause a problem though if the object is too big. For now, 100 will be okay.

- Storing the connections in a way That would be easy for me to go through the list of connection and make the change. I fixed this by –
    - Using set<pair<string, string>>. This way I can catch duplicates.
    - I can also put the smaller value as .first and the bigger value .second. This will make it easier to detect the bigger value and turn them into the smaller value via for-loop.
- The Object amount display the wrong amount because the last place the iterator is the next address within the array. This means the number of objects were always one more than it should be. I had fixed this by –
    - I got the integer value of the last place of the pointer and subtract 1 from it. `amount = stoi(*f) - 1`
    - For the second pass I took the amount and subtracting the size of the set of connections. `amount = amount - connections.size()`
- This final issue that I had ran into was being that two passes wasn't enough to rid the chart of extra 'larger' values. I fixed this by –
    - I made a while loop to detect whether there were any 'larger' values left. If it does detect the larger values, it will keep going until all those values are gone.

Data structures

In total, this project did not need to use very many data structures, the data structures that were used were all primitive. There were no additional objects/class made and/or used. I will list the data structure below and broadly explain what they were used for.

- fstream – This was used to open the file.
- string – General usage, user's answers, temps
- vector<string> - Used to make the rows
- vector<vector<string>> - Used to make the chart
- int – Used for the Object count
- bool – Used to check loops
- istringstream – Used to be able to use getline
- string * - Used to iterate the string array
- string [] - Used to hold the numbers for the Object count
- set<pair<string, string >> - Holding the pair of connecting Objects

Functionality

There in total I have five functions, not including main(). Those functions are -

- `string menu()`
- `bool fileReader(fstream &, string)`
- `vector<string> makeRow(string)`
- `void change(vector<vector<string>> & chart)`
- `void displayPass(vector<vector<string>> & chart, int)`

I will explain some of the smaller and more simple function briefly and the larger function more thoroughly below -

`string menu()` – This function is to get the user's file name. Once the input is complete the function will return the string. Main will then toss the string to fileReader along with an fstream.

`bool fileReader(fstream &, string)` – This function will read in the file and make a chart out of it. Once the chart is made it will give the chart to the change function. Its parameter is the fstream and the filename given by the user. Inside the function it calls the function makeRow to divvy up the responsibility of creating the chart. It will return true after it's done.

`vector<string> makeRow(string)` – This function will make the rows for the chart. The rows are in forms of a 1D vector. How it does this is by taking in the lines of the file and chop it up by delamination of a comma. This function will also replace all of the 1's into x's for easier changes later. Onion the 1D vector is made it will return the vector.

`void change(vector<vector<string>> & chart)` – This function is the main core of the program. It is called within the fileReader function after the chart has been made. Its parameter will be the 2D vector of string, which is the chart. Starting off the function will initialize the string array, string pointer, int amount, set<pair<string, string>> and int amount. Next a loop that will run 100 times to add values ("1" through "100") to the array of string. This will become its way of keeping track with the object count.

Next there will be a loop that goes through the whole chart. This will essentially be our first pass. We have a few special cases to avoid complications such as out of bounds –

- The first one is if we are in the first row. If we are then, are we in the first column? Both cases though – if we are in the first column or not - it will check if the position is an “x”. For the first column it will then become the first value in the array of strings. If it’s not the first column it will become the left’s side value IF it isn’t a “0”. If it is a “0” then it will be given a new object count number.
- The second case is if it isn’t the first row. First it will check if the current position is an “x”. Then if it isn’t the first row and if we are in the first column, we will look up to see if there is a value there. If there are no values above, it will give the position a new object count number. If the it isn’t the first row and it isn’t the first column, we can then check the top AND left. The checking begins by seeing if either of the top or bottom are “0”. If they are then it knows to give the value of the position that isn’t “0” to the current position. Otherwise if they are containing any not “0” see what they are. This will then compare the two values and assign the current position to the lesser. At this point, it will make the pair of strings (connections) that will be used to change chart. The lesser will be the ‘key’ value. If both values are indeed zero, then the currently will have no choice but to be assigned the next new object count value.

Once the chart is done with the first pass, we will output the chart using the displayPass function along with the number of objects there are.

For the second pass and beyond, there will be a while-loop. The while loop is to make sure that all of the ‘larger’ object values are changed into the ‘smaller’ object value. It does this by changing all the ‘larger’ object value to the ‘smaller’ object value. It then set done to true to break out of the loop UNLESS if any of the ‘larger’ values are still within the graph, it will set done back to false to go back and finish the job. This case is when there are more than 2 numbers attached to each other i.e. 1→2 & 2→3.

Finally, it calls the displayPass function giving the function the new chart and the amount of object minus the size of the set of connections.

`void displayPass(vector<vector<string>> & chart, int)` – This function will just display the given chart and the given object amount. This function was called in the change() function.

The program will replay if need be. This will happen in main.

Share of the work

I had worked alone on this project. I've always hated doing project alone because catching mistakes can be a bit rough. Although this was tough it beats What I had to do last project which was pairing up with someone that doesn't know how to code let alone C++.