TaskInterview

compiler clang++-9

std c++2a

Explain why I choose vector:

I implemented the class using vectors, I used vectors because they are "familiar" objects in C ++, allows copying, deleting & iterators already built into the language. Also to expand the above class, I will add that using a vector allows you to access all the algorithms built into C ++ by sending Begin & End and create iterators without self-realization. However vectors take up more space on memory than a standard array that takes up exactly the space of assignment.

Explain the class :

The class I built is a generic class that can accept any object, In the implementation I created, I added class-unique operators such as - , [ ] , == , to make it easier for the user to compare, subtract, and access values in a particular location. However if you want to use operettas for self-realization, the typename that the class receives must have the operettas -, ==.

Makefile:

To make it easier for the user with compile, I built a Makefile that builds the program and returns two executables the first is a demo and the second is test The first is a file showing the user how to use the class and the second is an executable of the tests including a score of how many passed and how not.

```
make
./test // for run test
./demo // run demo
```

Algorithm (draw_line)

I use the Bresenham algorithm ,why is use the bresnham algo? I used this algorithm because it solves the line problem by progressing with integers and is the most efficient algorithm O (abs (y2-y1)) when y2 > y1. The algorithm does not solve the whole problem but a small part of it, because in a task it is required to draw a line in any direction, and from any point to any point. So I expanded the algorithm to work for any line on the matrix, whether from top to bottom, or "go" backwards ,When I performed the tests I realized that when the density is very small, the algorithm does not work related so I built functions that "help the algorithm" with a small density and "rotate" the matrix and perform it differently so that it fits any density. How it work ? : First the algorithm checks which sub-algorithm to go, by slope, then for each sub-algorithm a calculation is made so that there will be progress whether to progress (x, y + 1) or (x + 1, y + 1) according to the epsilon value , The algorithm starts with an error (eps) equal to 0, then adds the value of Dx or Dy depending on the type of slope, and checks which way to proceed And checks whether eps * 2 is greater than the distance x or the distance y depending on the type of slope

Methods

`void draw_vertical_line(int x1, int y1, int x2, int y2, T value)`

- explain : Draws inside the image a straight line lengthwise or widthwise
- output: draw vertical line

`void draw_std_lower_case(int x1, int y1, int x2, int y2, T value)`

- explain : A function that produces a line between two points when m(slope) is negative
- output: draw line with negative slope

`void draw_rotate_lower_case(int x1, int y1, int x2, int y2, T value)`

- explain := Draws a straight line between two points, when the slope is negative, and when the points x1, x2 are far from eachother and y1, y2 are close and the density is small, the function "rotate" the matrix and calculates according to a larger density
- output : draw line with negative slope and small density

`void draw_std(int x1, int y1, int x2, int y2, T value)`

- explain := Draws a line from point to point when the slope is positive and uses a change of Bresenham Line-Drawing Algorithm
- output : draw line with positive slope

`void draw_rotate(int x1, int y1, int x2, int y2, T value)`

- explain :=Draws a straight line between two points, when the slope is positive, and when the points x1, x2 are far from eachother and y1, y2 are close and the density is small, the function "rotate" the matrix and calculates according to a larger density
- output : draw line with positive slope and small density

`int check(int x1, int y1, int x2, int y2)`

- explain := Checks which algorithm (postivie slope) performs better density and returns FLAG to that algorithm
- output : flag of the algorithm

`int check(int x1, int y1, int x2, int y2)`

- explain := Checks which algorithm (negative slope) performs better density and returns FLAG to that algorithm
- output : flag of the algorithm

`T& operator[](std::pair<int,int> location)`

- explain := get pair<int,int> and return & to the value in that point
- output : value in that point

`T operator[](std::pair<int,int> location) const`

- explain := get pair<int,int> and return & to the value in that point
- output : value in that point

`picture& operator-(picture<T>& to_delete)`

- explain := Gets an image & and subtracts an image from an image by subtracting a value from the first by value in the right pic, saves the result in the first image from which you subtract
- output : this->_pic - to_delete

`bool operator==(picture<T>& check_equal)`

- explain :=Performs a comparison between two images returns true if they are equal
- output : true if (check_equal == _pic)

`void draw_line(int x1, int y1, int x2, int y2, T value)`

- explain :=The main algorithm that draws to every side side of the algorithm breaks down into sub-cases and sends to the most appropriate case according to the values it received
- output : draw line any direction

`void displayPic()`

- explain :=Prints the matrix of the image on top of the terminal, to be used there must be a load for special classes(Pixel) of the ostream operator
- output : print matrix on terminal

Tests (check its correctness)

To check if the program works, I built a test system, using open source. I built a number of functions that check if there is a line between the given points.

the main function:

`bool check_the_draw(picture<T> & r ,int x1 ,int y1,int x2,int y2)`

- explain : The function starts from (x1, y1) and checks in each iteration whether there is a number around the number we are on, deletes the number we are on and continues to the next number, and so on until it reaches (x2, y2) if we could not reach (x2, y2) This is a sign that the drawing we made did not work and the line was not created correctly between the two points and that means the slope was "too deep" and the number failed to reach the target point
- output : true/false if have line between 2 points and if the draw_line work properly.

`pair<int,int> check_if_around(picture<T> & r,int x,int y)`

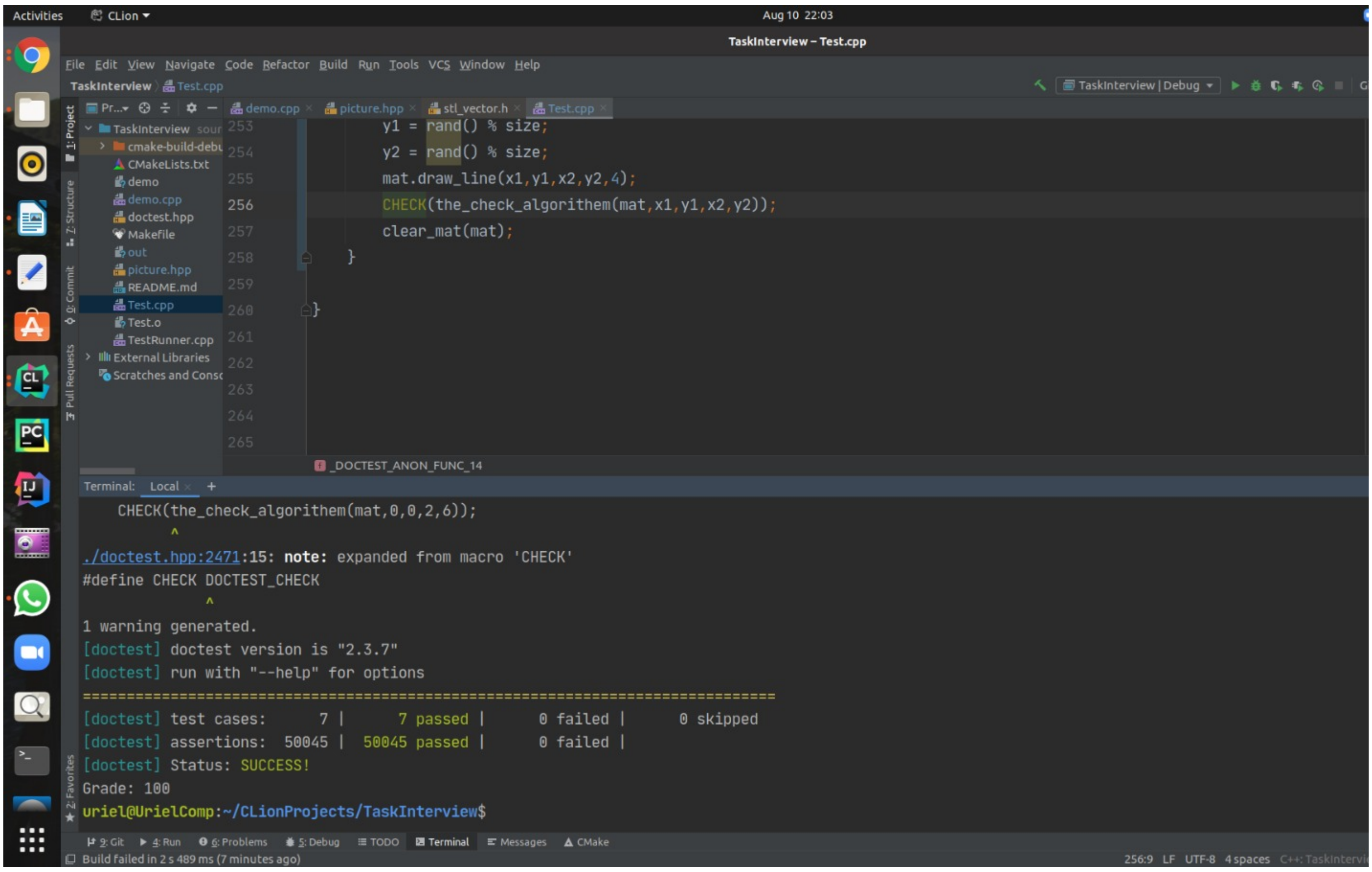
• explain : A function that receives a pair and an image and checks whether around the number we are checking (y-1, y + 1, x-1, x + 1) whether there is a number that is a continuation of the line of the same number, otherwise the function returns the pair (-1, -1) That in the main function it will return a false * output : pair<int,int> of the next value on the line

`void clear_mat(picture<T> & r)`

- explain : A function that in each test resets the matrix so as not to go over values that have already been given output : clean matrix

Explain doctest.h :

doctest.h is open source, which allows you to perform an assert (CHECK in doctest) of all kinds, check for throws exception , check for a false or true return, or compare two objects, At the end of the tests, a printout is made to the terminal of whether tests failed, how many succeeded and what type of test failed and why. Example :



```
            y1 = rand() % size;
            y2 = rand() % size;
            mat.draw_line(x1,y1,x2,y2,4);
            CHECK(the_check_algorithem(mat,x1,y1,x2,y2));
            clear_mat(mat);
        }

}
```

```
    CHECK(the_check_algorithem(mat,0,0,2,6));
          ^
./doctest.hpp:2471:15: note: expanded from macro 'CHECK'
#define CHECK DOCTEST_CHECK
              ^
1 warning generated.
[doctest] doctest version is "2.3.7"
[doctest] run with "--help" for options
===============================================================================
[doctest] test cases:      7 |      7 passed |      0 failed |      0 skipped
[doctest] assertions:  50045 |  50045 passed |      0 failed |
[doctest] Status: SUCCESS!
Grade: 100
uriel@UrielComp:~/CLionProjects/TaskInterview$
```

Division into cases of tests :

Case 1 : Checking whether an exception is thrown when bad values are entered into a function

Case 2 : Checking whether an exception is thrown when different values or external classes are entered in the constructor

Case 3 : Check draw_line when the slope is positive

Case 4 : Check draw_line when the slope is negative

Case 5 : Check when there is a straight line lengthwise and widthwise

Case 6 : In this case, an automatic test of 100,000 random tests is performed, when 4 random numbers (x1, y1, x2, y2) are selected, and a random image size is selected, and in each iteration the numbers are checked by the test function, and in each case a test is performed, at the beginning of writing the code I started with 20 iterations and so on, until I discovered the mistakes and improved the algorithm so that it would work for any random case.

Sources

- https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
- doctest