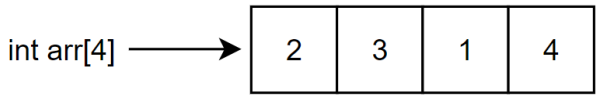
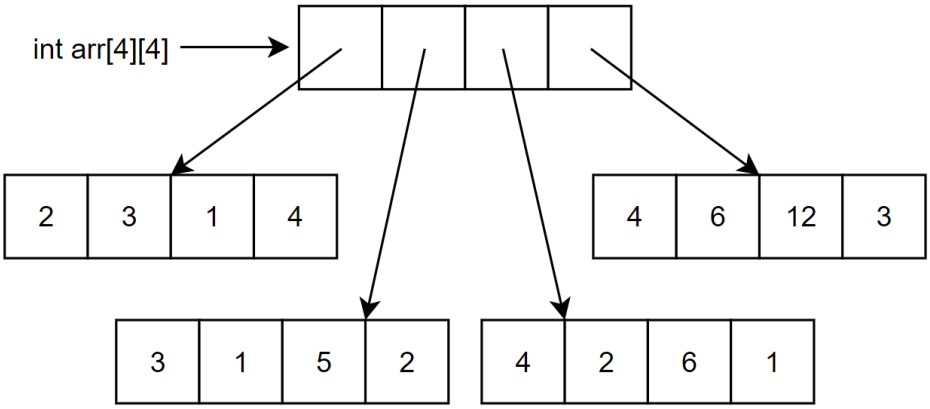
SDL2 Course Day-4
Map Object

2D Array:

A one dimensional array can be represented in memory like this:



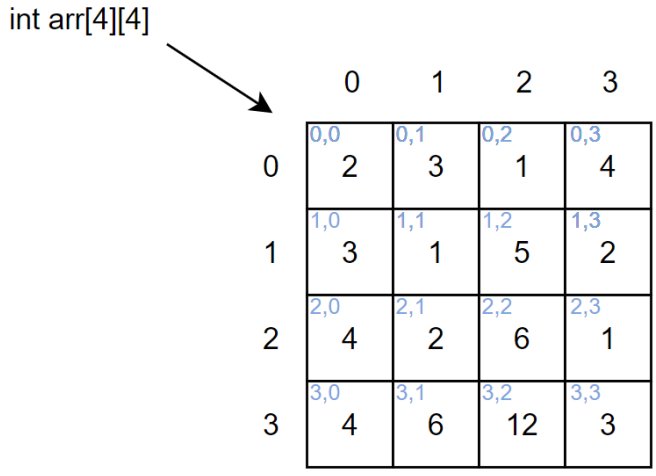
A two dimensional array is basically an **array of arrays** and can be represented like this:



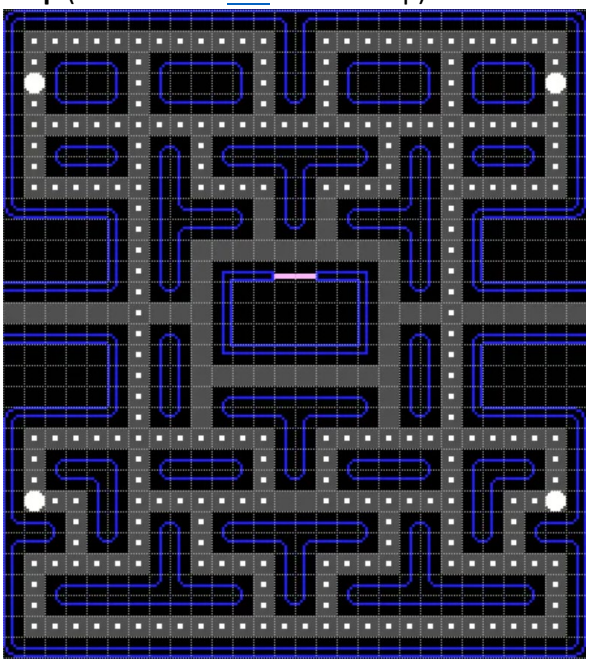
In Code we create it like this:

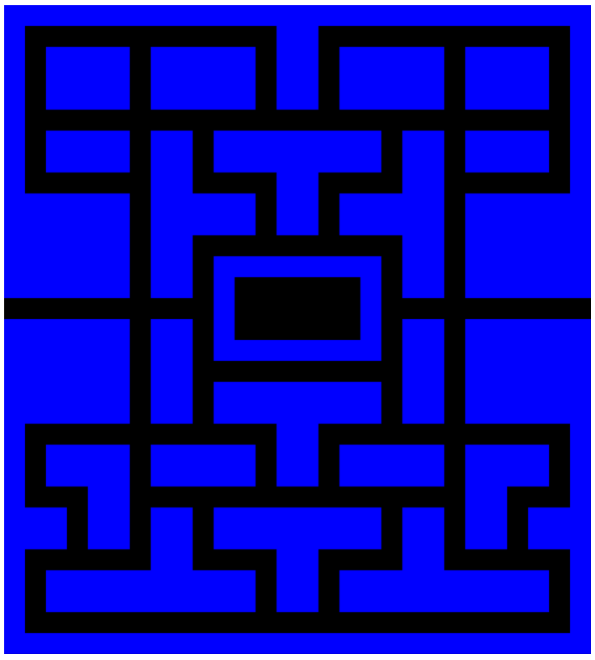
```
int arr[4][4] = {
        {2, 3, 1, 4},
        {3, 1, 5, 2},
        {4, 2, 6, 1},
        {4, 6, 2, 3}
};
```

For our benefit we imagine a 2D array like this:



Map (Refer to this link for the map)**:**





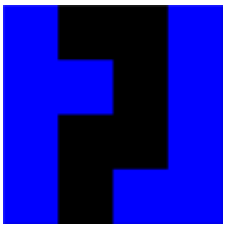
As we can see that our map contains many walls and to represent the walls in our game (without images for now), we would need to create the background on the right. However, if we did this manually, we would have to spend a lot of time trying to manually find out where to place the rectangle and the dimensions and this would be very painful.

Instead, if we look at the image on the left, the map is designed in such a way that it can be represented in the form of a 2D array in which if the number is 1, it's a wall and if its anything else, the player can move in this region. We call this as a tile-map (since we represent the map as a collection of tiles).

For now we will work with a basic tile-map to get the hang of it:

```
int tile_map[4][4] = {
        {1, 0, 0, 1},
        {1, 1, 0, 1},
        {1, 0, 0, 1},
        {1, 0, 1, 1}
};
```

=====>



(1 indicates wall, 0 indicates path)

Creating the Map Object:

Like our other game objects the map objects can have handle_input, update and draw methods. However, our map will not change if the any event happens (if the user presses any key, our map is not affected), so there is no need for the map to have a handle_input function. Hence the map object will only have a constructor and draw methods. To display the map we would need to know the width of each cell so we pass in as parameters to the constructor the cell width and the cell height. And like other objects we also pass in the renderer to save a copy of it in our object.

Hence our map_object.h will look as follows:

```
#ifndef SDL2_TEST_MAP_OBJECT_H
#define SDL2_TEST_MAP_OBJECT_H

#include <SDL2/SDL.h>

class Map_Object {
public:
    SDL_Renderer *Renderer;

    Map_Object(int cell_w, int cell_h, SDL_Renderer *Renderer);

    void draw();
};

#endif //SDL2_TEST_MAP_OBJECT_H
```

We would also need to store the number of rows and columns our tilemap contains and the tilemap itself. Once we have the tile-map we would also need to create rectangles and we would need a place to store them so that they can be accessed later (For this we use a vector). So we make the following additions:

```
#include <vector>

class Map_Object {
public:
    int rows = 4;
    int cols = 4;
    int tile_map[4][4] = {
            {1, 0, 0, 1},
            {1, 1, 0, 1},
            {1, 0, 0, 1},
            {1, 0, 1, 1}
    };
    std::vector<SDL_Rect> walls;
```

Now in our main.cpp file we can include our map object's header, create an instance of the map object, call its draw function inside the while loop and then delete it at the end of the program.

```
#include "Map_Object.h"
```

```
Map_Object *map = new Map_Object(25, 25, win_obj->Renderer);
```

Inside our while loop:

```
// drawing
win_obj->clear_background();  // clears background
map->draw();                  // draws map
player->draw();               // draws player
win_obj->swap_buffer();       // swaps buffers
```

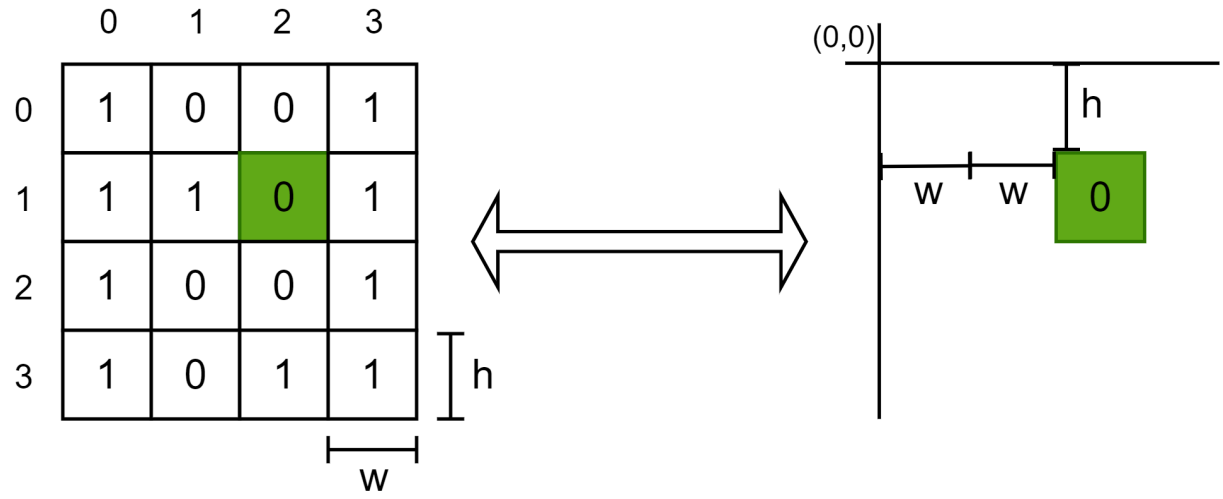
At the end of the program:

```
delete map;
```

Note that if we call the draw function of the player before the map, then the player would appear under the map (try it out).

Now in our map_object.cpp we iterate through are tile-map and if the element is 1 then we create a rectangle and then add it to our walls which is a vector.

Note that to create a rectangle from the array we would need to convert the array index to window pixel co-ordinates.



Notice that an index of (1, 2) on the tile-map is equivalent to (2 * cell_width, 1 * cell_height) on the window. This can be generalized to any tile of index (A, B) in the tile map is equivalent to (B * cell_width, A * cell_height) on the window screen.

Hence now we iterate through our tile-map and create a rectangle with the x and y position according to the above formula and then push it to the walls vector. Then in our draw function, we iterate through the vector walls. Refer to our github Day-4/Map_Object.cpp for the rest of the code.

Collision:

Refer to the following for Collision detection: http://kishimotostudios.com/articles/aabb_collision/
In SDL2, the above method is implemented under the function

```
SDL_HasIntersection(&rect1, &rect2)
```