Tilemaps

First, define the size of the tilemap in number of tiles across (width) and number of tiles down (height).

```
int mapWidth = 7;
int mapHeight = 5;
```

Now, define an array of tile identifier values, one value for each tile in the tilemap.

In the example below, the value `1` will represent a wall tile, and the value `0` will represent a floor tile.

```
int map[] = {
    // map tile ids
    1, 1, 1, 1, 1, 1, 1,
    1, 0, 0, 0, 0, 0, 1,
    1, 1, 1, 0, 0, 0, 1,
    1, 0, 0, 0, 1, 0, 1,
    1, 1, 1, 1, 1, 1, 1,
    //
};
```

Let's print out the tilemap to the console using a period character for the floor tile, and a hashmark character for the wall tile like this:

```
#######
#.....#
###...#
#...#.#
#######
```

There are at least two ways that will achieve this output.
Here is the nested for-loop method:

```
// loop over every row of tiles down the tilemap
// starting from the 0th at the top, down to
// the (mapHeight - 1)th row at the bottom
for (int y = 0; y < mapHeight; y++) {
  // loop over every column of tiles across the tilemap
  // starting from the 0th at the left, across to
  // the (mapWidth - 1)th column at the right
  for (int x = 0; x < mapWidth; x++) {
    // calculate the index value to
    // read from the tilemap array. this is a
    // simple calculation of the column plus the
    // row times the width of the tilemap.
    int index = x + (y * mapWidth);

    // with the index, get the tile id at the
    // corresponding location of the tilemap.
    int tileId = map[index];

    // now draw the tile
    char tileCharacter = '?';
    if (tileId == 1) {
      // wall
      tileCharacter = '#';
    } else if (tileId == 0) {
      // floor
      tileCharacter = '.';
    }
    std::cout << tileCharacter;
  }
  // at the end of the row, draw a new line
  std::cout << std::endl;
}
```

Now here is a single for-loop method for the exact same output.

```
// loop over every single tile in the tilemap
// starting from the 0th at the top left,
// across from the left to the right,
// then down to the next row, back at the left
// repeating this until reaching the end of the tilemap
int tileCount = mapWidth * mapHeight;
for (int index = 0; index < tileCount; index++) {
  // the loop is providing the index that we saw
  // in the previous example.
  // calculate the column from the index
  // by taking the remainder after dividing the
  // index by the width of the tilemap
  int column = index % mapWidth;
  // calculate the row by dividing the index
  // by the width of the map and truncating the
  // decimal. using integer math, this will truncate the
  // decimal automatically. If floating point math was
  // used, the decimal would need to be "floored".
  int row = index / mapWidth;

  // with the index, get the tile id at the
  // corresponding location of the tilemap.
  int tileId = map[index];

  // now draw the tile
  char tileCharacter = '?';
  if (tileId == 1) {
    // wall
    tileCharacter = '#';
  } else if (tileId == 0) {
    // floor
    tileCharacter = '.';
  }
  std::cout << tileCharacter;

  // if the current column is the last column
  // in the row, then draw a new line
  if (column == mapWidth - 1) {
    std::cout << std::endl;
  }
}
```

With that out of the way, the basics of the tilemap have been explained.

Continue reading to learn how to turn the simple text based tilemap output from above, into a graphical example using SFML.

SFML Tilemap

Setup the vertex array.

```
// vertex array that will hold the vertices of the tilemap
sf::VertexArray mapVerts;
// the quad primitive means that 4 vertices per tile are needed
mapVerts.setPrimitiveType(sf::PrimitiveType::Quads);
// resize the vertex array to hold all vertices needed
mapVerts.resize(4 * mapWidth * mapHeight);
```

Loop over the tilemap data in single for-loop method explained before.

```
for (int i = 0; i < mapWidth * mapHeight; i++) {
  int x = i % mapWidth;
  int y = i / mapWidth;
  int tileId = map[i];
  // using the same logic, find the correct tile
  // x (U) and y (V)
  int tileU = tileId % numTilesAcrossTexture;
  int tileV = tileId / numTilesDownTexture;
  // get a pointer to the current quad
  sf::Vertex *quad = &mapVerts[4 * (x + y * mapWidth)];
  // set the positions for each vertex
  // there are four vertices for each tile
  // the first vertex is the top left of the tile
  // the second vertex is the top right of the tile
  // the third vertex is the bottom right of the tile
  // the fourth vertex is the bottom left of the tile
  // this defines the vertices in clockwise order and is
  // important to do so for proper rendering

  // the tile size is multiplied by the coordinates to position
  // in the correct location on the screen, if the size of the
  // tile was not multiplied, then the tilemap would render at
  // a 1:1 scale wth the size of the tilemap, in the example
  // above, that is 7x5 which is quite small.
  // however, if the tile size were 96x96 pixels, then rendering
  // the tilemap would be 96 * 7 by 96 * 5 which is 672x480 and
  // that is what is done below.
  float left = x * tileWidth;
  float top = y * tileHeight;
  float right = (x + 1) * tileWidth;
  float bottom = (y + 1) * tileHeight;
  quad[0].position = sf::Vector2f(left, top);
  quad[1].position = sf::Vector2f(right, top);
  quad[2].position = sf::Vector2f(right, bottom);
  quad[3].position = sf::Vector2f(left, bottom);

  // once the positions of the vertices are set, then the texture
  // coordinates for the vertices need to be set. texture coordinates
  // are represented by U and V values.
  // normally UV coordinates are normalized and fall within
  // the range 0.0 and 1.0 however, in SFML, they are given
  // as the pixel offsets into the texture.
  // this simplifies the calculations to be the same as the positions
  // with the x and y replaced by the u and v
  float leftU = tileU * tileWidth;
  float topV = tileV * tileHeight;
  float rightU = (tileU + 1) * tileWidth;
  float bottomV = (tileV + 1) * tileHeight;
  quad[0].texCoords = sf::Vector2f(leftU, topV);
  quad[1].texCoords = sf::Vector2f(rightU, topV);
```

```
  quad[2].texCoords = sf::Vector2f(rightU, bottomV);
  quad[3].texCoords = sf::Vector2f(leftU, bottomV);
}
```

All that is needed is to draw the tilemap vertex array with the desired texture to the window.

```
window.draw(mapVerts, &tilemapTexture);
```

And that is all there is to it!