The Game-Of-Life stream module shown below connects the Game-of-Life array to the display to construct the Game of Life.

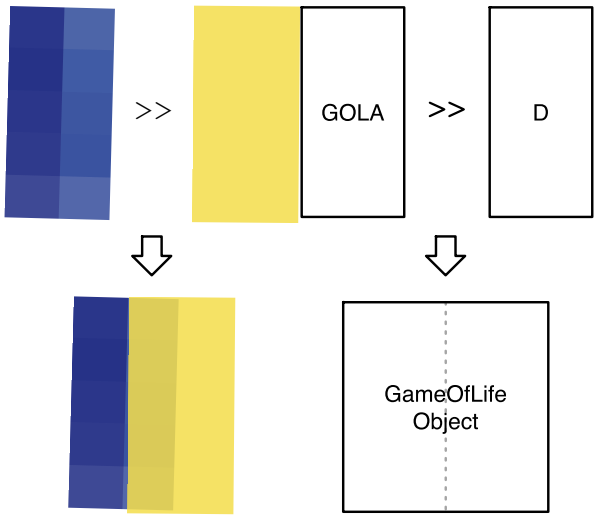
```
class GameOfLife: public streamModule
{
    GameOfLifeArray GOLA; // GameOfLife object
    display         D;    // display object

public:

    GameOfLife()          // Constructor
    {
        GOLA >> D;        // Connect outputs of GOLA to inputs of D
        end();            // Housekeeping
    }
};
```

Some observations about GameOfLife:

a. *Lego® blocks* are a world-wide phenomenon due to their ability to create arbitrarily complex structures by simply *snapping blocks together* as illustrated in Figure 1(a). The TruStream module operator >> has a similar ability: With it, a programmer can create arbitrarily complex TruStream structures by simply *snapping modules together*. In our case, snapping two modules together means connecting the outputs of one module to the inputs of the other module.



(a) Lego® Blocks (b) TruStream Modules

Figure 1. Snapping Blocks and Modules Together

b. The module operator >> has two alternate prototypes:

```
module module::operator >> ( module );
module module::operator << ( module );
```

and two corresponding forms:

```
M1 >> M2
M2 << M1
```

where M1 and M2 are modules (either thread or stream). In both cases, the *visible* output streams of module M1 are connected to the *visible* input streams of M2.

c. In the GameOfLife constructor, the statement

```
GOLA >> D;
```

connects the nRowsXnCols output-stream array of GOLA – arising from the output streams of the cell–module array in GOLA – to the nRowsXnCols input-stream array of D as illustrated in Figure (b).

d. And that completes construction of the TruStream program for the Game-of-Life cellular automaton.