Computer Engineering and Mechatronics MMME3085

Dr Louise Brown

Software Engineering Best Practice

Part 1

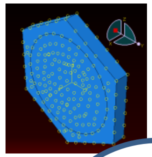
What is Software Engineering?

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"— IEEE *Standard Glossary of Software Engineering Terminology*

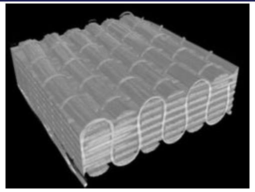
Covers the whole process required to produce a software product

TexGen Geometric Textile Modelling Software

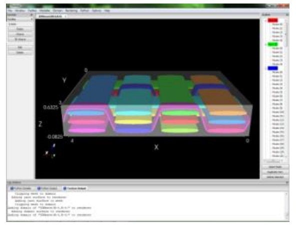
Fibre/Micro-Scale



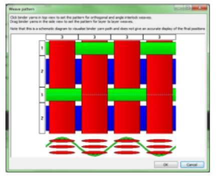
Micro-scale FEA simulations or analytical methods determine yarn properties



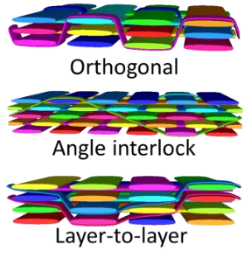
Unit Cell/Meso-Scale

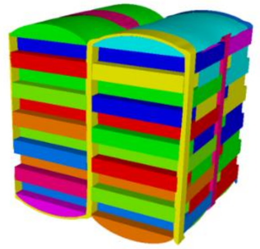


Generate textile geometry using TexGen GUI or script

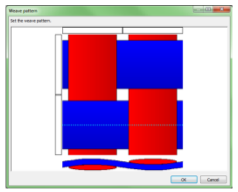


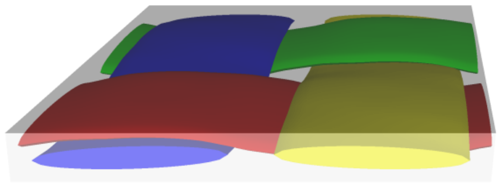
3D wizard generates idealised 3D textiles



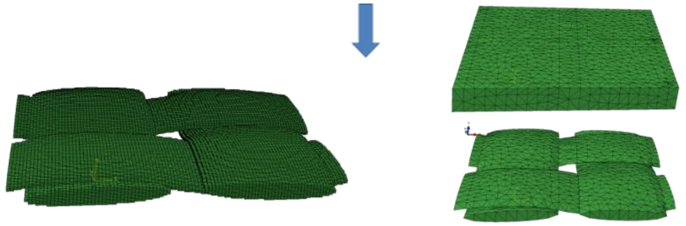


Refinement of orthogonal weave to simulate compaction





Automatically generate 2D and 2D sheared textiles



Generate mesh and input files for FEA or CFD to predict material properties

Composite material properties extracted from meso-scale predictions are used to model structural components

Component/Macro-Scale

What is involved in creating software?

What are the things you need to consider to create a piece of software and/or a software product?

What are the steps in the process?

These were your ideas at the start of the module – any changes?

https://padlet.com/louisebrown7/overview-of-a-software-project-ttiklf86efk760zq

Overview of a Software Project

What’s involved in creating a piece of software?

Requirements gathering

High level design

Low level design

Development

Testing

Deployment

Maintenance

Overview of a Software Project

What's involved in creating a piece of software?

Requirements gathering

High level design

Low level design

Development

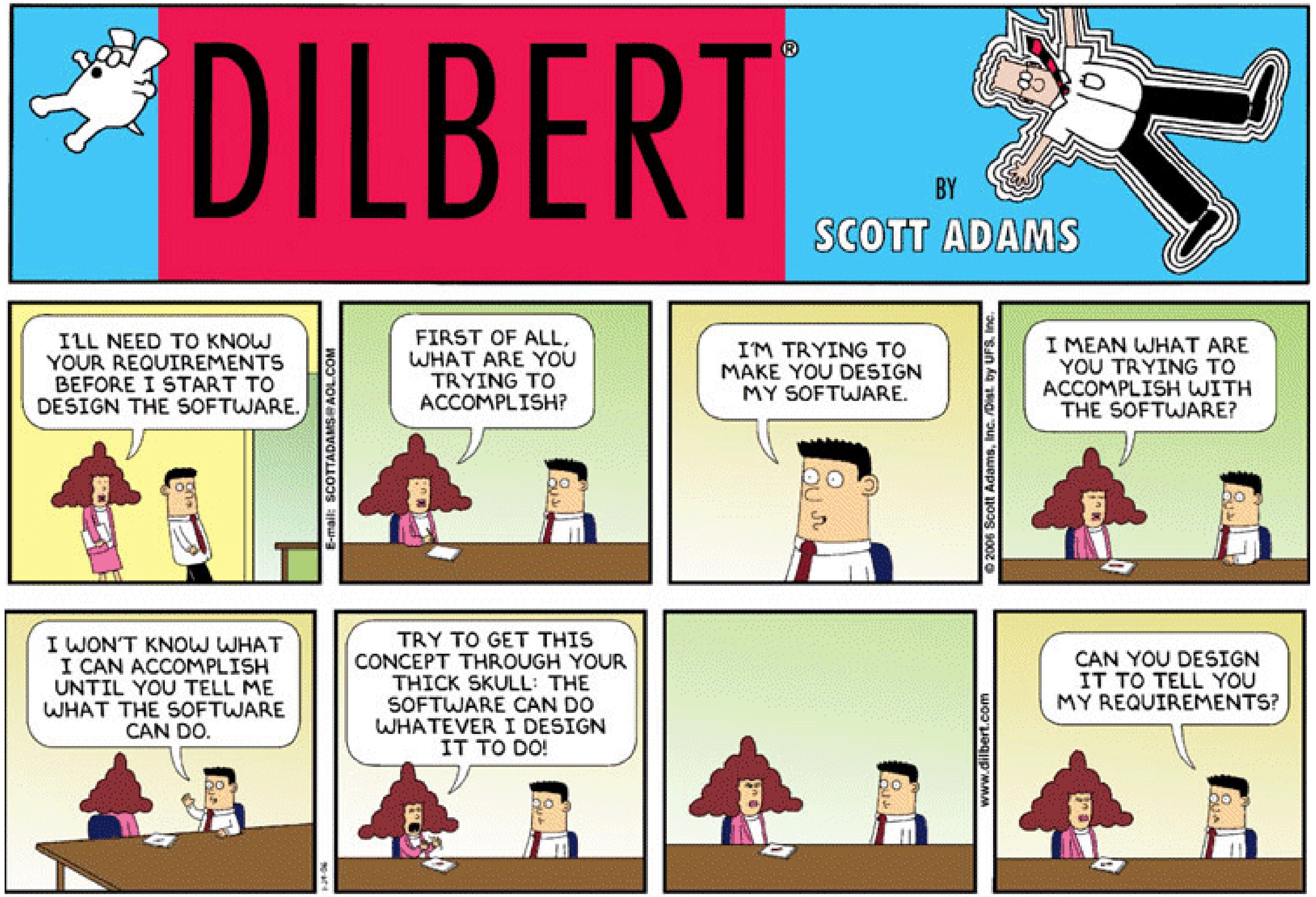
Testing

Deployment

Maintenance

Requirements Gathering

Dilbert
by Scott Adams


DILBERT®
BY SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
E-mail: SCOTTADAMS@AOL.COM
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
© 2006 Scott Adams, Inc. /Dist. by UFS, Inc.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
www.dilbert.com
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?

Requirements Gathering

Good design starts with being able to define your problem (in language your user can understand)

If you can't explain something to a six-year-old, you really don't understand it yourself – Albert Einstein

Specify the *requirements* – what features your software must provide

Must be precise, clear and unambiguous

Prioritise – what are the essentials and which are 'nice to have'

Verifiable – can it be tested that the requirement has been met?

Robot Writing Project

In line with the design processes covered in the course you will be required to produce a specification document using the template on Moodle (ProjectPlanningTemplate23-24.docx):

- **A specification of <u>precisely</u> what the program needs to do**
- The forms of the data stored within your program
- The planned function declarations (prototypes) for each function identifying whether parameters are input, output or changed, and return value if any. You are encouraged to give a return value which indicates successful execution or failure.
- Test cases for each function to confirm conformance of the function to its specification.

You could try writing this first to see if you've understood the problem

You need also to provide a flowchart showing the operational flow of your code.

Overview of a Software Project

What's involved in creating a piece of software?

Requirements gathering

High level design

Low level design

Development

Testing

Deployment

Maintenance

High Level Design

Gives an overall view of a system

Defines the major components of a system and their interactions. These can be thought of as a set of building blocks each with its own set of responsibilities. Communication rules between blocks should be well defined.

Specify major classes and data. Think about why a specific data format or file type is to be used. Consider any libraries which can be used.

User interface design. This should not affect the classes and data already specified.

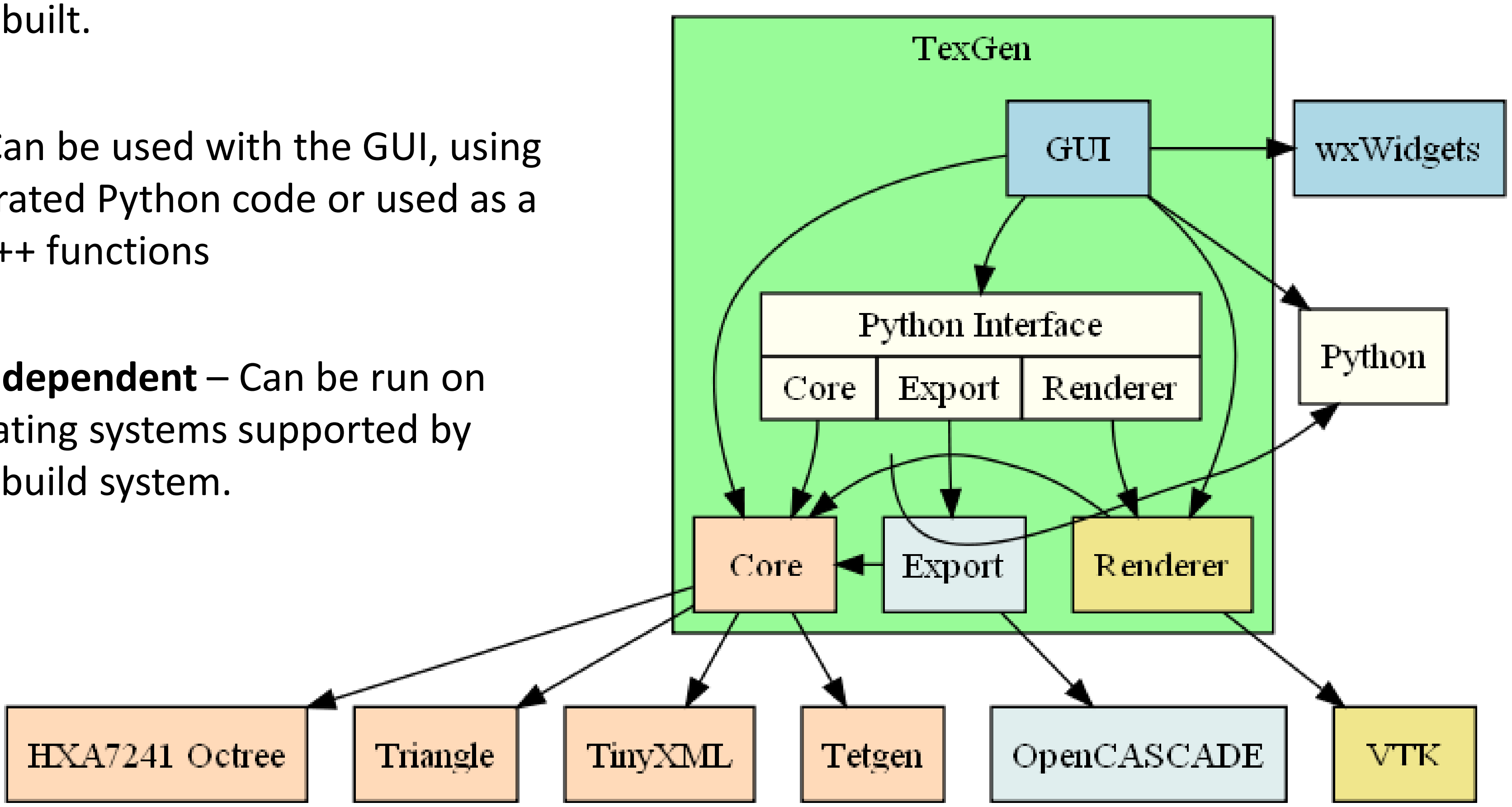
May use tools such as UML (Unified Modelling Language)

High Level Design

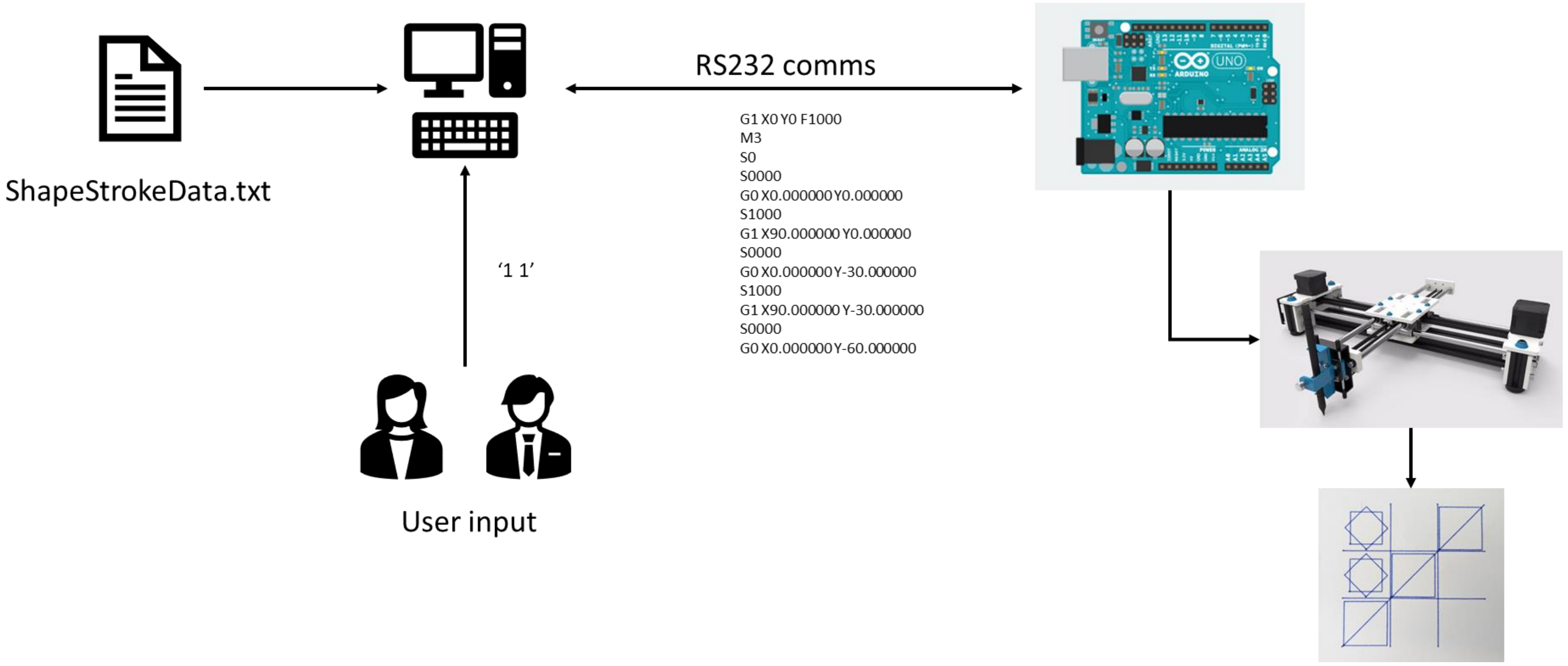
Modular -Core functionality is in the core module, graphics are in a renderer module; if not using visualisation, the renderer doesn't need to be built.

Flexible – Can be used with the GUI, using SWIG generated Python code or used as a library of C++ functions

Platform independent – Can be run on most operating systems supported by the CMake build system.



Project High Level Design



```
G1 X0 Y0 F1000
M3
S0
S0000
G0 X0.000000 Y0.000000
S1000
G1 X90.000000 Y0.000000
S0000
G0 X0.000000 Y-30.000000
S1000
G1 X90.000000 Y-30.000000
S0000
G0 X0.000000 Y-60.000000
```

Overview of a Software Project

What’s involved in creating a piece of software?

Requirements gathering

High level design

Low level design

Development

Testing

Deployment

Maintenance

Low Level Design

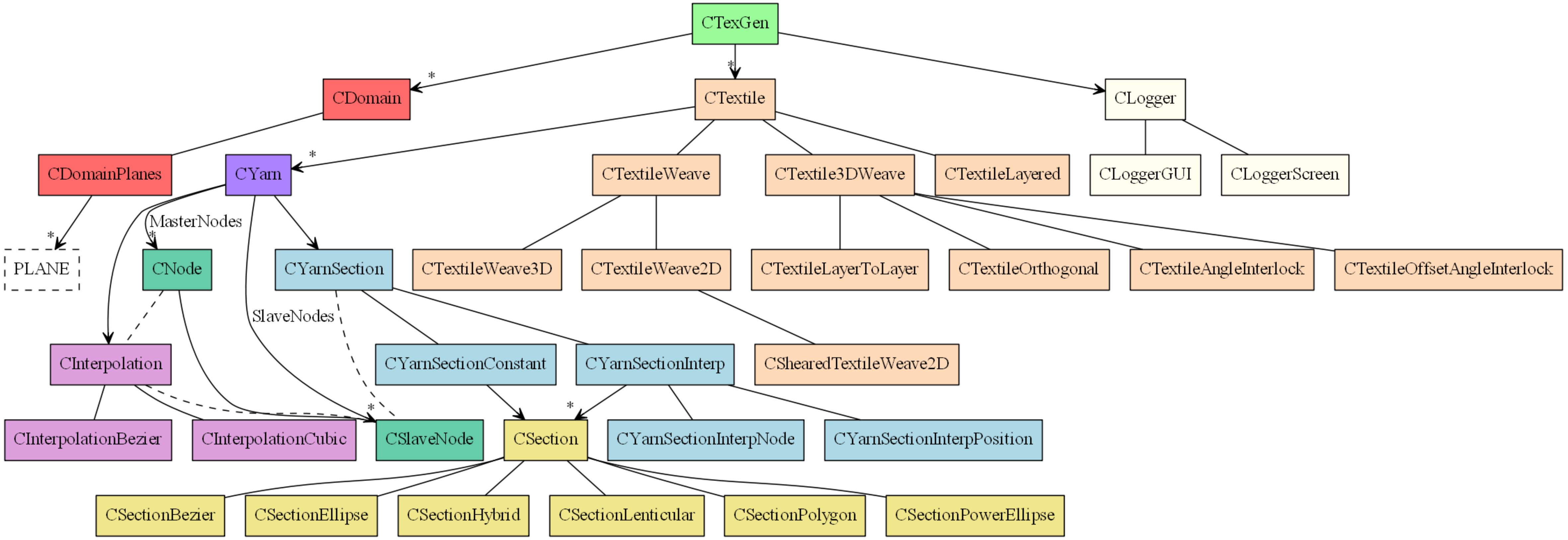
Provides the detail about how the high level design will be implemented.

Don’t dive into the detail straight away. Start to refine the detail of what functions will do, what classes or data structures are needed.

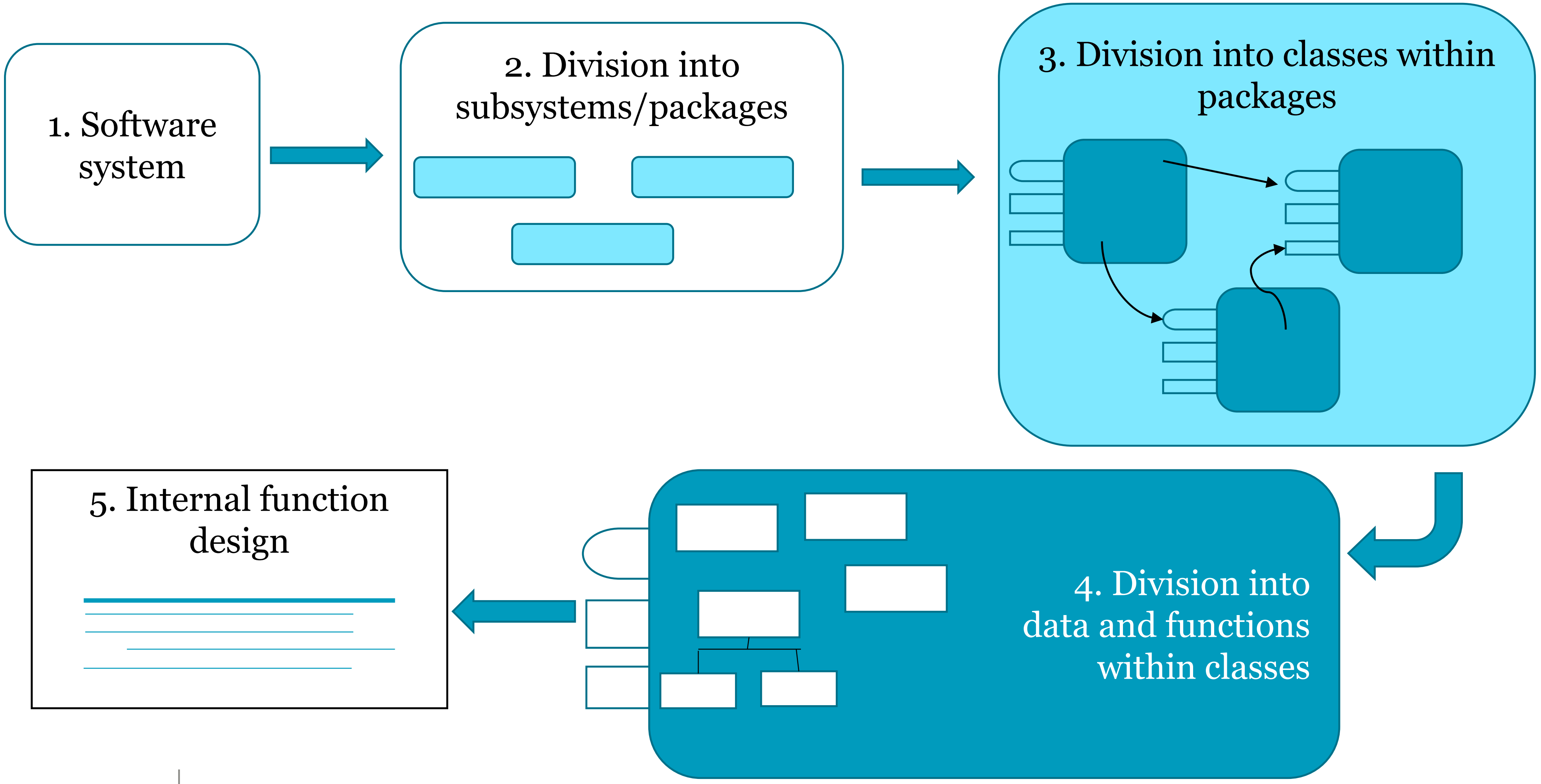
Define the interface – what is passed in and out of a function, what parameters can be changed

This can be an iterative process. For example if several functions all pass the same set of parameters it may be that these should be grouped together in a structure so the data structure may need to be revisited.

TexGen Core Class Heirarchy



Levels of Design


1. Software system
2. Division into subsystems/packages
3. Division into classes within packages
4. Division into data and functions within classes
5. Internal function design

Robot Writing Project

In line with the design processes covered in the course you will be required to produce a specification document using the template on Moodle (ProjectPlanningTemplate23-24.docx):

- A specification of precisely what the program needs to do
- **The forms of the data stored within your program**
- **The planned function declarations (prototypes) for each function identifying whether parameters are input, output or changed, and the return value if any. You are encouraged to give a return value which indicates successful execution or failure.**
- Test cases for each function to confirm conformance of the function to its specification.

You need also to provide a flowchart showing the operational flow of your code.

Appendix 1

Splitting code across multiple files
(not in the book)

Building better projects (1)

All the example code provided has been written to a single file which also contains main

This is not ‘bad’ but it does create restrictions if we wish to ‘distribute’ code development as there is only one master source file

It also means that if we wish to reuse code then we have to copy/paste it from one program to another

- Not ideal if you have ‘common’ code used across a number of applications
- It also means if we come up with a ‘better’ way to do things we have to the repeat the process in all our applications

Building better projects (2)

To get round this, we look to ‘split’ code into a number of files, each (say) containing code to perform a specific set of tasks

If we then need to develop a new application that needs this functionality we can simply ‘add’ this code to our project

To do this is not complex however there are a few things we need to do

Building better projects (3)

For each new code (.c) file you plan to use you will also need to pair this with a header (.h) file.

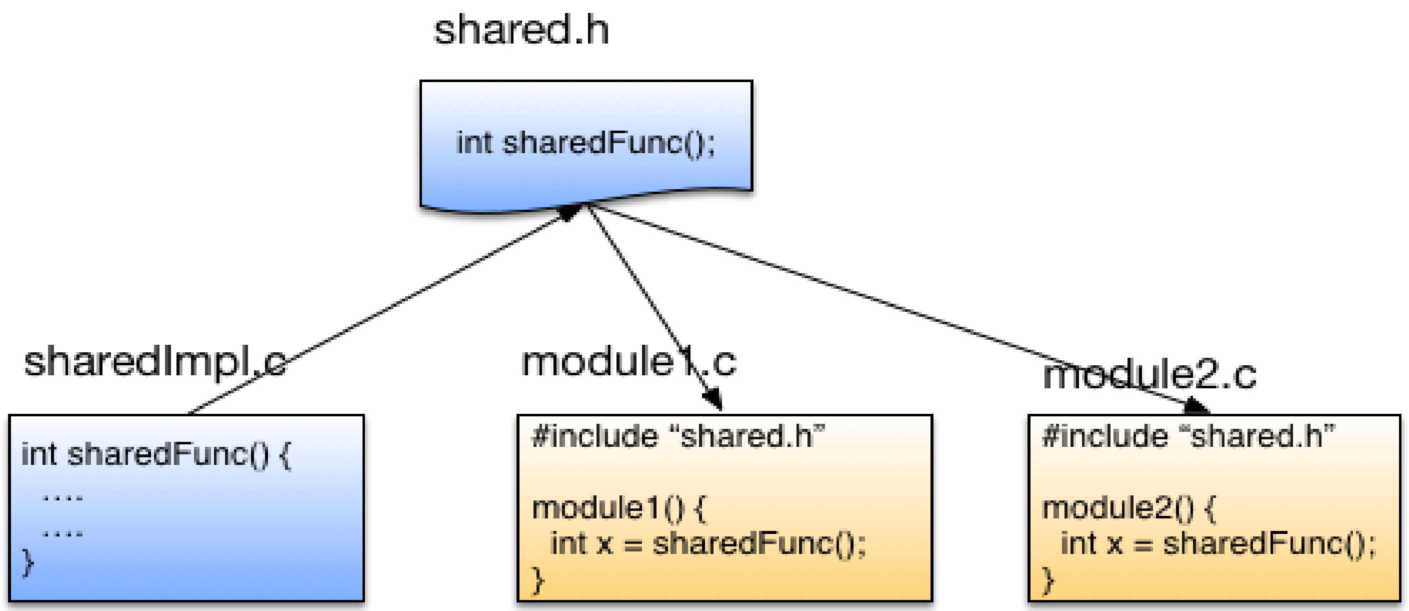
- The .c file contains the lines of ‘working’ code,
- The .h, header file, contains information about the functions provided – you should also place in here any #define statements needed.

The ‘.c’ file you create will need to #include its own header file – plus any others it itself needs (for example stdio.h, stdlib.h & possibly other header files that you have created).

In other files that you wish to make use of the functions defined in the source code (.c) file created you must include the header (.h) file.

Code and header files

This is shown graphically below



```
int sharedFunc();
```

```
int sharedFunc() {
  ....
  ....
}
```

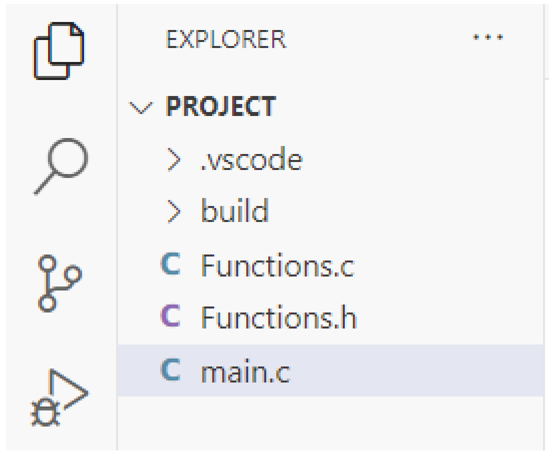
```
#include "shared.h"

module1() {
  int x = sharedFunc();
}
```

```
#include "shared.h"

module2() {
  int x = sharedFunc();
}
```

http://hanxue-it.blogspot.com/2014/04/why-include-cc-implementation-code-in.html

Doing this in practice… (1)

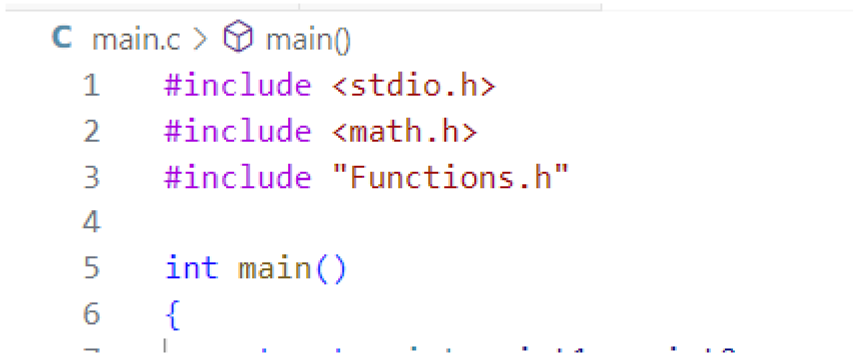
- Start by creating a folder for your project
- Add a .c file which will contain the main() function to the folder
- Add a .c and a .h file for the files which will contain the functions



Doing this in practice… (2)

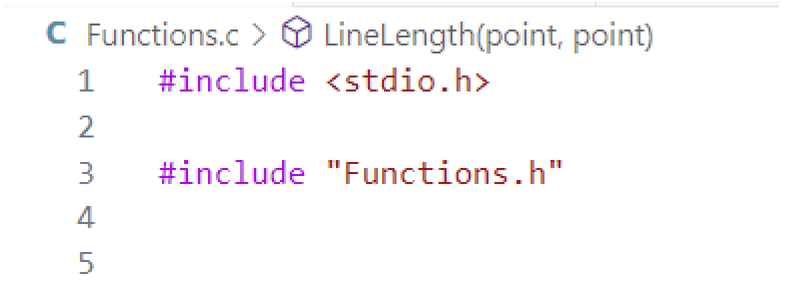
Start writing your code remembering…

- To include all the 'standard' include files in each (if you need them)
- The header file you have just created
  - Note the use of "" instead of <> in the #include – this is to let the compiler know the include files are 'local' to your application



main.c > main()

```c
#include <stdio.h>
#include <math.h>
#include "Functions.h"

int main()
{
```



Functions.c > LineLength(point, point)

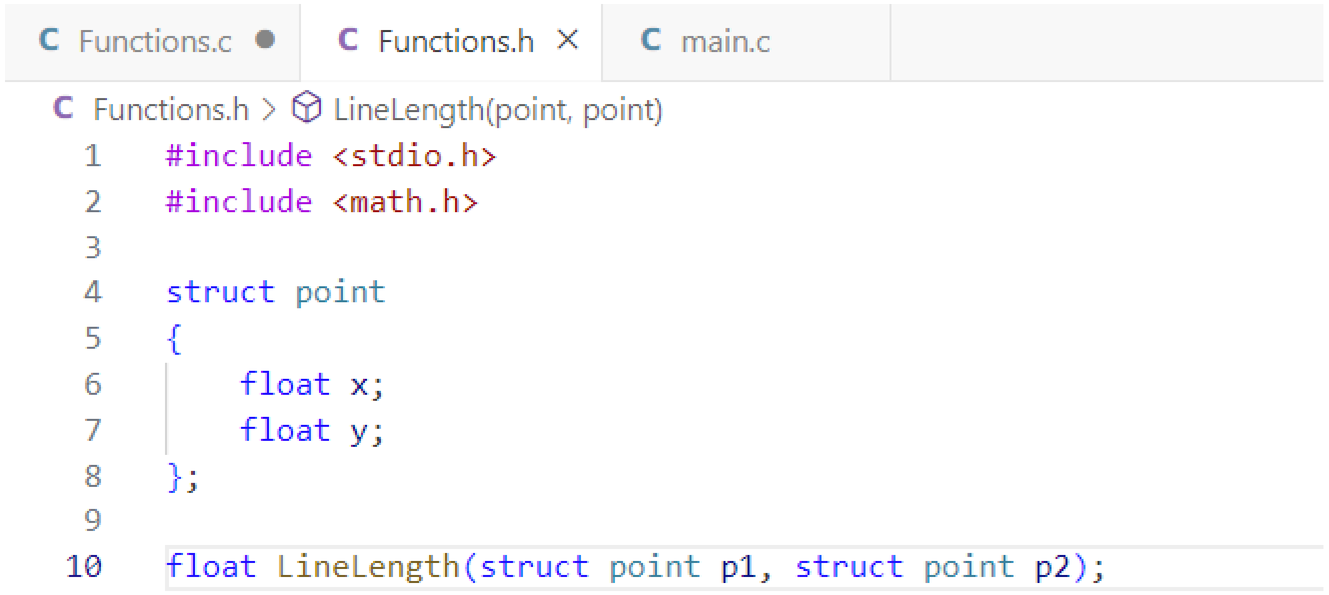
```c
#include <stdio.h>

#include "Functions.h"


```

Doing this in practice… (3)

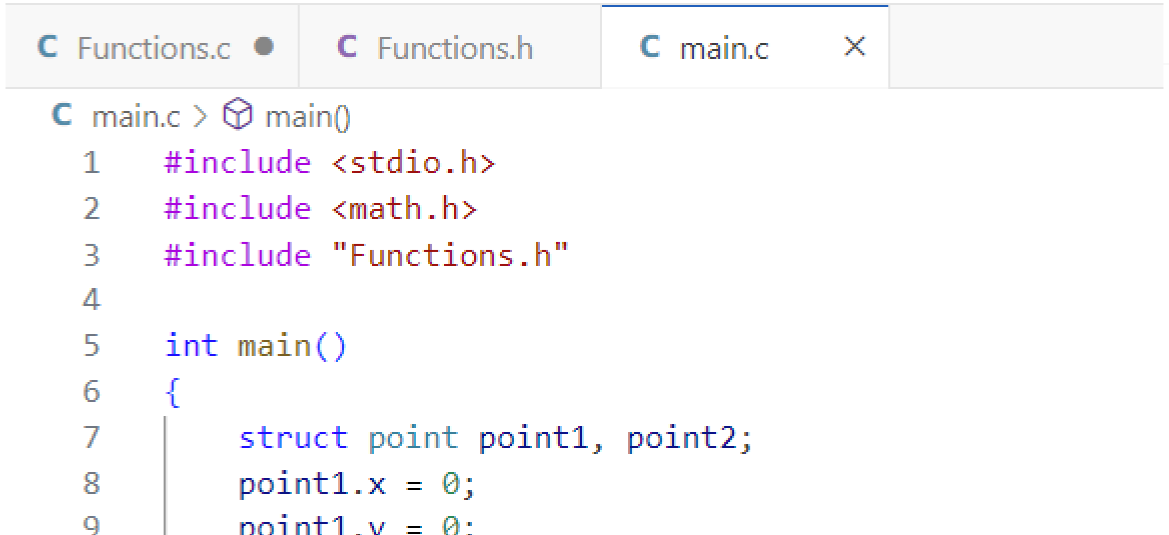
- Now develop your code putting things in the ‘correct’ files
- Any function created in a file that you wish to allow other files to use you need to create a header for in the .h file
- We might also add some #defines

Giving us ……

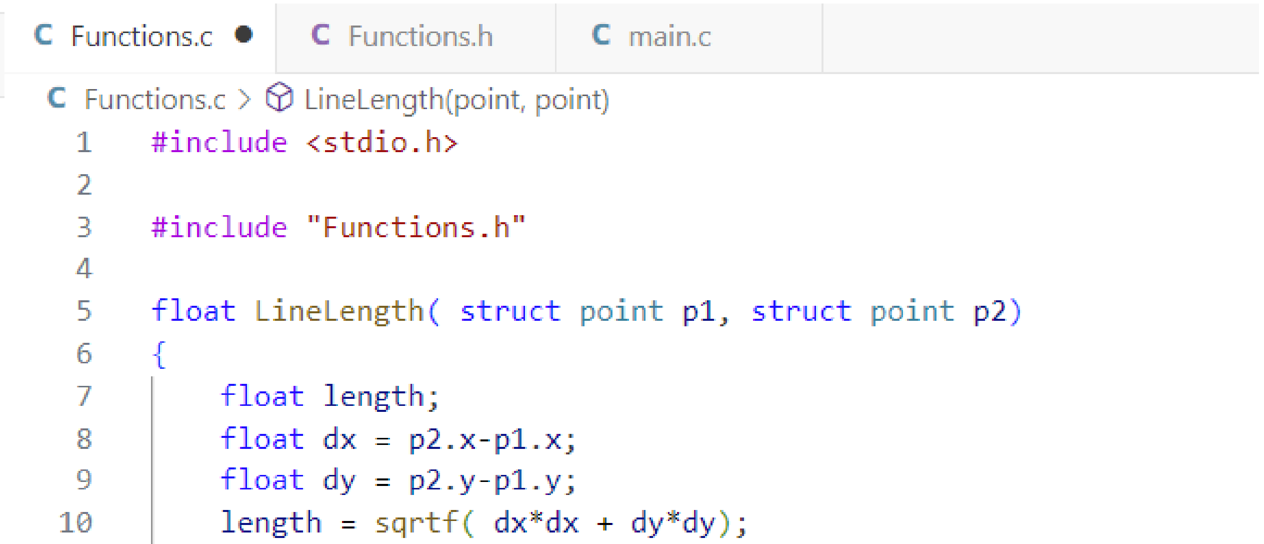
Doing this in practice… (4)



```c
#include <stdio.h>
#include <math.h>

struct point
{
    float x;
    float y;
};

float LineLength(struct point p1, struct point p2);
```



```c
#include <stdio.h>
#include <math.h>
#include "Functions.h"

int main()
{
    struct point point1, point2;
    point1.x = 0;
    point1.y = 0;
```



```c
#include <stdio.h>

#include "Functions.h"

float LineLength( struct point p1, struct point p2)
{
    float length;
    float dx = p2.x-p1.x;
    float dy = p2.y-p1.y;
    length = sqrtf( dx*dx + dy*dy);
```

Software Project

As an aside (useful for the project!)

There is an equivalent sprintf which prints to a string, this can also be very useful

```
char buffer[100];
int age;
/* code to assign variables….. */
sprintf ( buffer, "my name is %s, age %d ",name, age)
printf(“%s”,buffer);
```

There is also a version that restricts the number of characters ‘printed’ to the specified string

- Good to avoid overrunning the end of your string (and so avoid program crashes)

```
snprintf ( buffer, 100, "my name is %s, age %d ",name, age)
```

Limits to buffer size specified here, will truncate (if required) to this size

G-Codes

G-Code is the programming language used by CNC (Computer Numerical Control) machines.

It stands for ‘Geometric code’

We are using a limited subset of the codes:

Use G0 for rapid positioning with the pen up

Use G1 when drawing with the pen down

Command	Description
F1000	feed rate (i.e. pen speed) 1000 mm min^{-1}
G0 X Y	Move to the position X,Y
G1 X Y	Draw a straight line from the last position to X,Y
M3	Turn on Spindle (needed for arm to work!)
S0	Pen up (original meaning is ‘spindle speed 0’)
S1000	Pen down (original meaning is ‘spindle speed 1000 rev min^{-1}’)

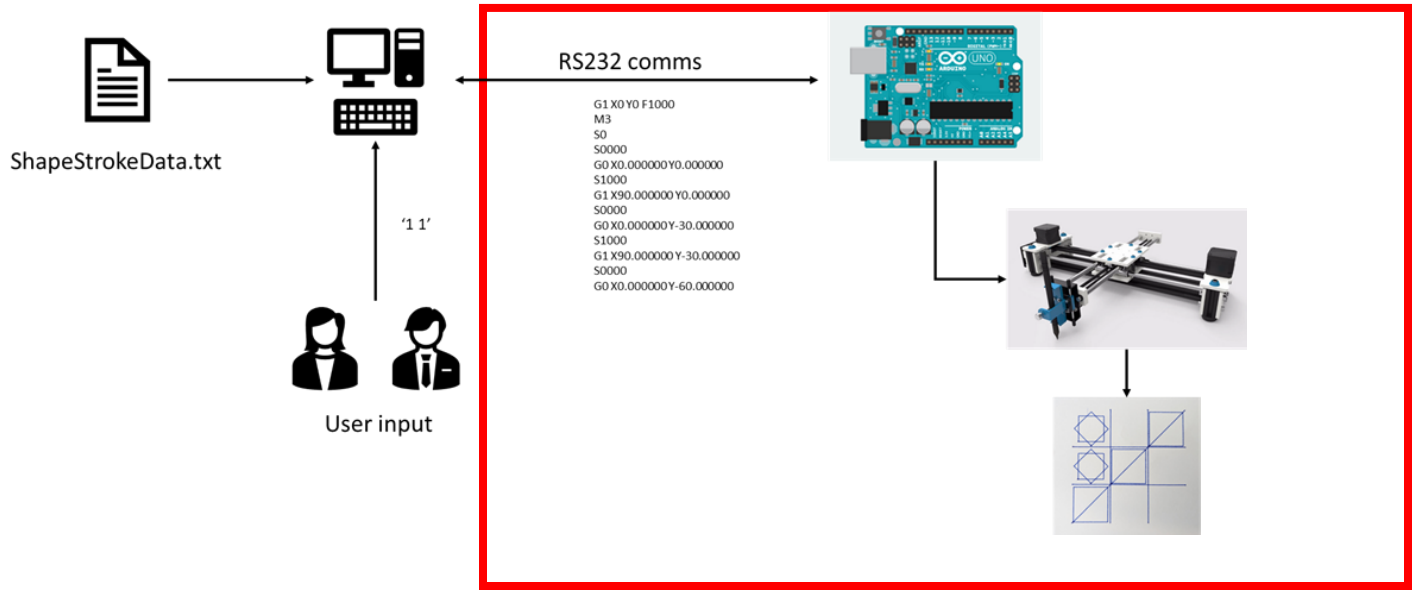
There is a nice blog post here: https://howtomechatronics.com/tutorials/g-code-explained-list-of-most-important-g-code-commands/

Sending G-code to Arduino and awaiting acknowledgement – Appendix 4

A virtual serial port is used to send the G-Code commands

An RS-232 library written by Teunis van Beelen is used.

The sample code in RobotWriter5.0.zip on Moodle gives a sample project for sending some hard-coded G-code



```
G1 X0 Y0 F1000
M3
S0
S0000
G0 X0.000000 Y0.000000
S1000
G1 X90.000000 Y0.000000
S0000
G0 X0.000000 Y-30.000000
S1000
G1 X90.000000 Y-30.000000
S0000
G0 X0.000000 Y-60.000000
```

The Serial.c file uses a #ifdef statement to either send the G-code to the serial port or to be printed (to enable testing using the emulator)

Another look at the Serial Communications!

Open SerialEchoBlink.ino

This reads in a string of characters and sends an ‘ok’ message and toggles the LED when a new line character is reached.

This can be seen by running the program with the Serial Monitor open.

At this point serial communications via the COM port are with the Serial Monitor.

Another look at the Serial Communications! (2)

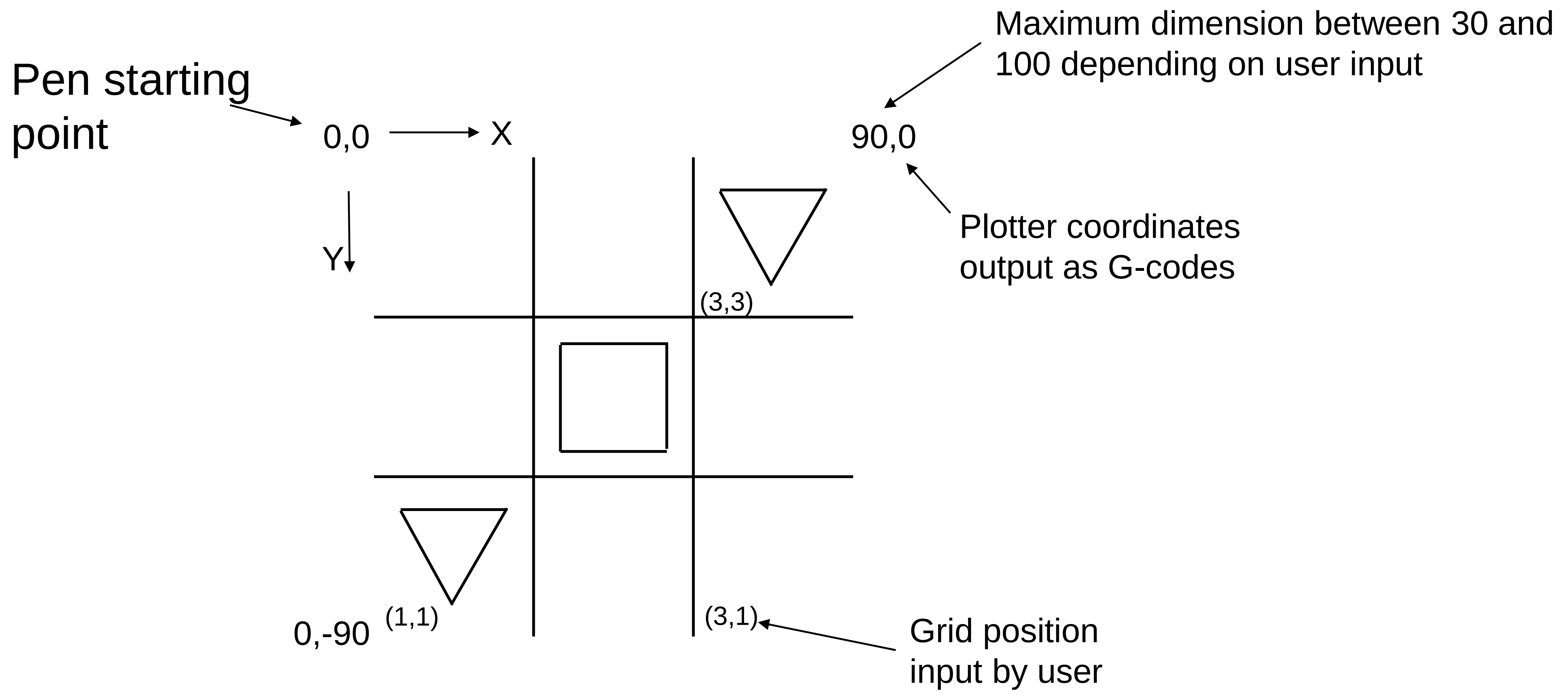
Keep SerialEchoBlink.ino running but close COM port.

Open and run the main function in the RobotWriter5.0_Skeleton project.

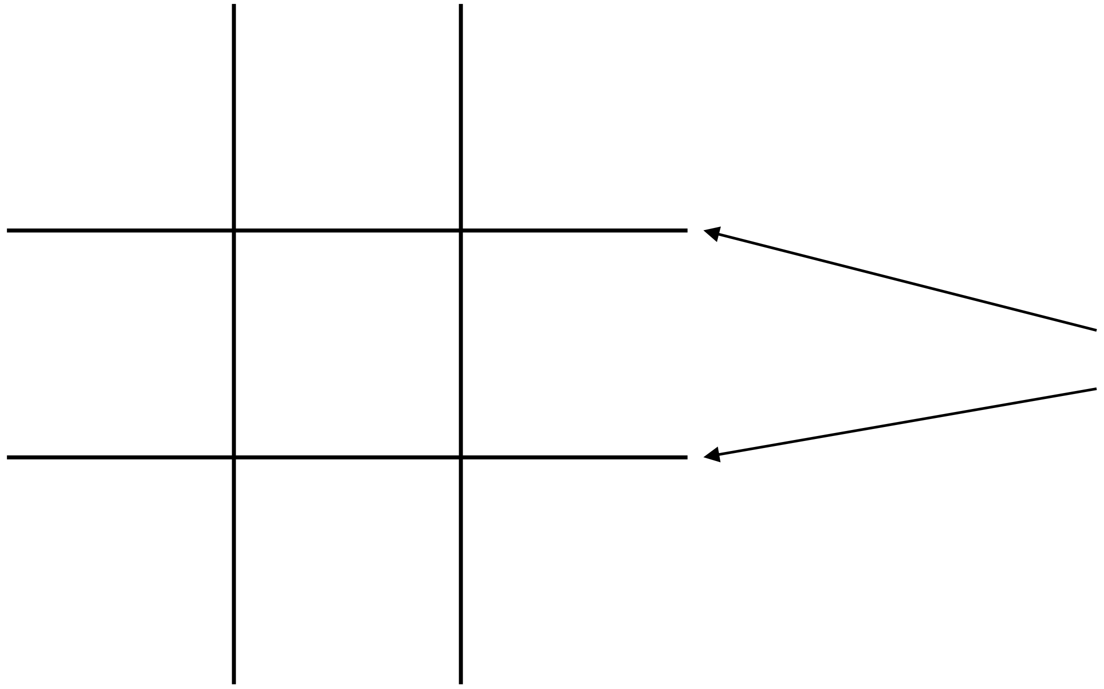
The main() function calls functions in Serial.c which open, close, read and write to the specified COM (serial) port.

These, in turn, call the lower level functions in the rs232 library which implement the actual serial communications with the port.

Robot coordinates


Maximum dimension between 30 and 100 depending on user input
Pen starting point
0,0
X
90,0
Y
(3,3)
Plotter coordinates output as G-codes
0,-90
(1,1)
(3,1)
Grid position input by user

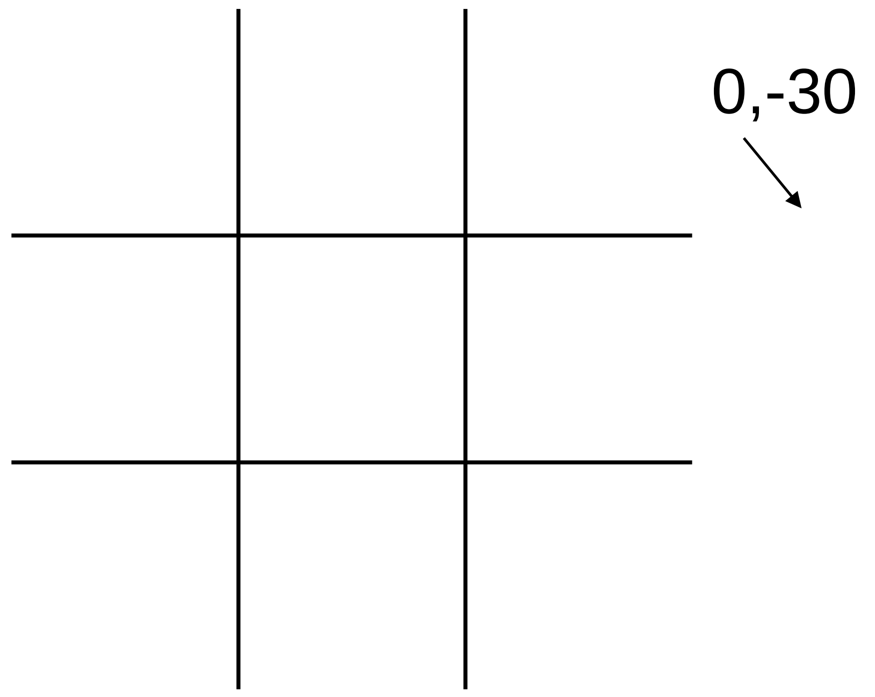
Drawing the grid



Decide which order to draw lines

Example – drawing the two horizontal lines

Drawing the grid – size 90mm (1)

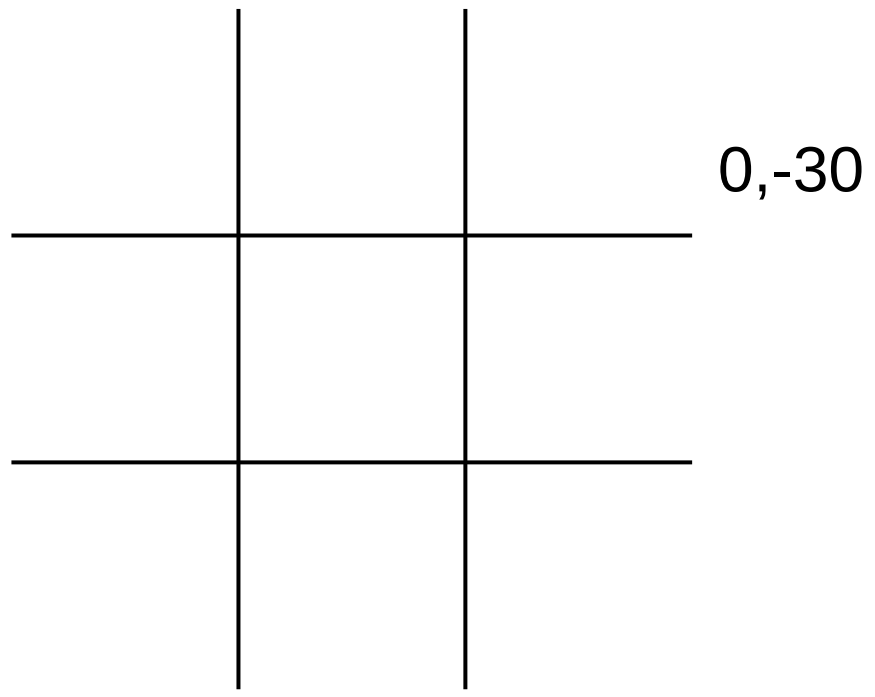


S0000 ← Pen up

G0 X0.000000 Y-30.000000

Move to 0,-30

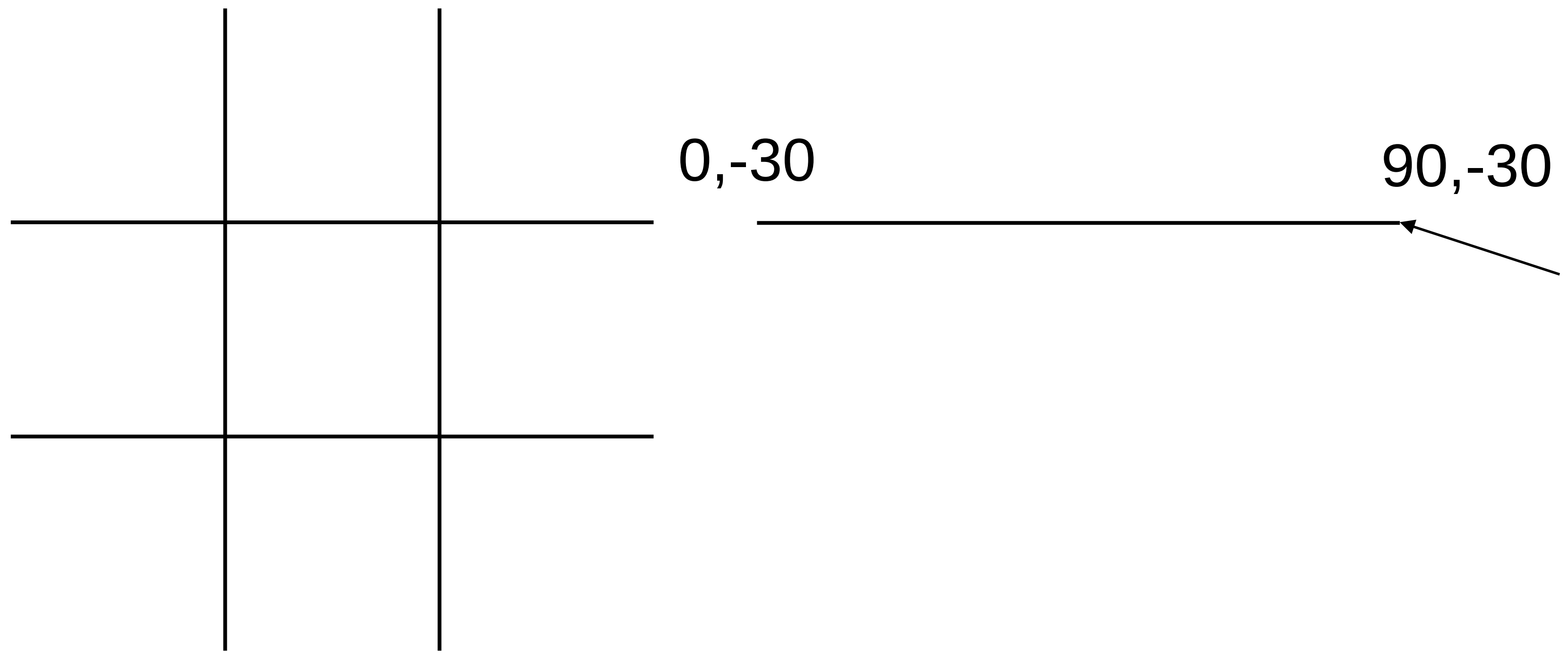
Drawing the grid – size 90mm (2)



```
S0000

G0 X0.000000 Y-30.000000

S1000
```

S1000 ← Pen down

Drawing the grid – size 90mm (3)

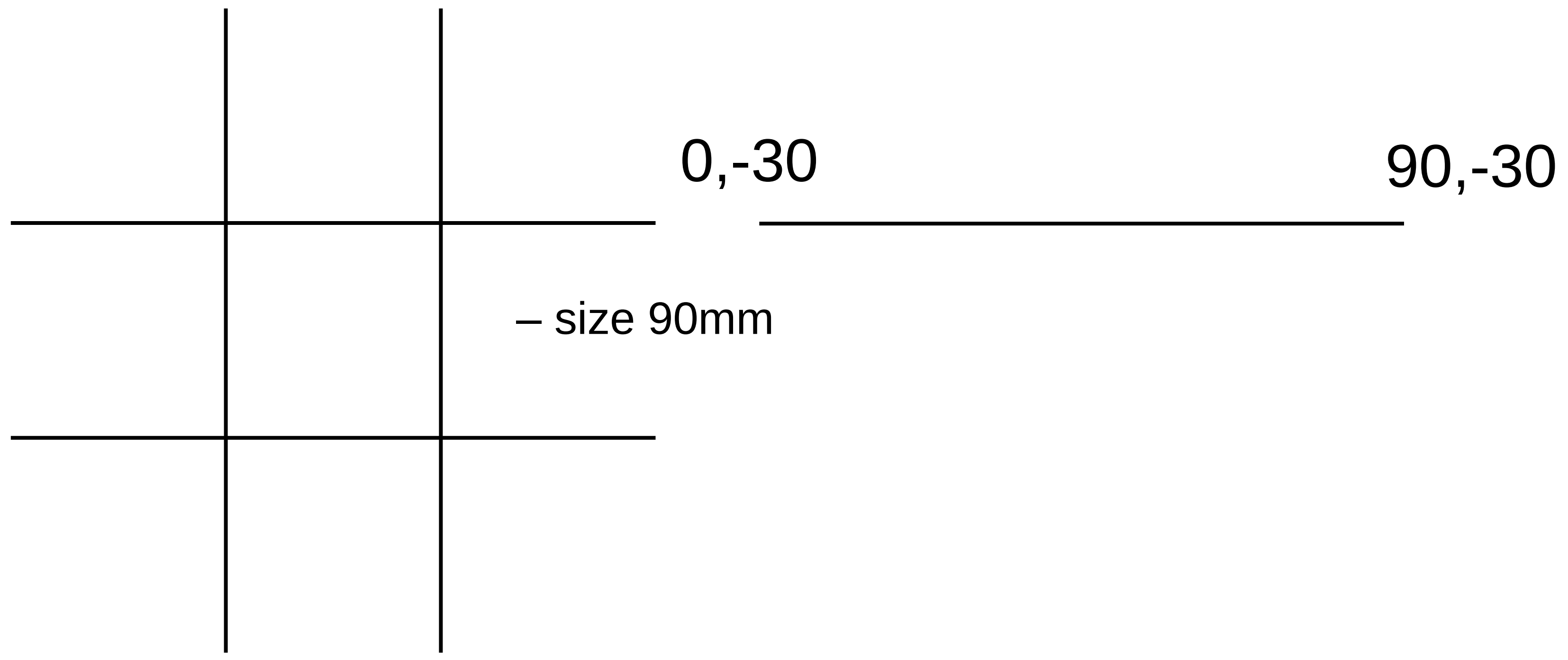


```
S0000

G0 X0.000000 Y-30.000000

S1000

G1 X90.000000 Y-30.000000
```

Pen down

Move to 90,-30

Drawing the grid – size 90mm (4)



S0000

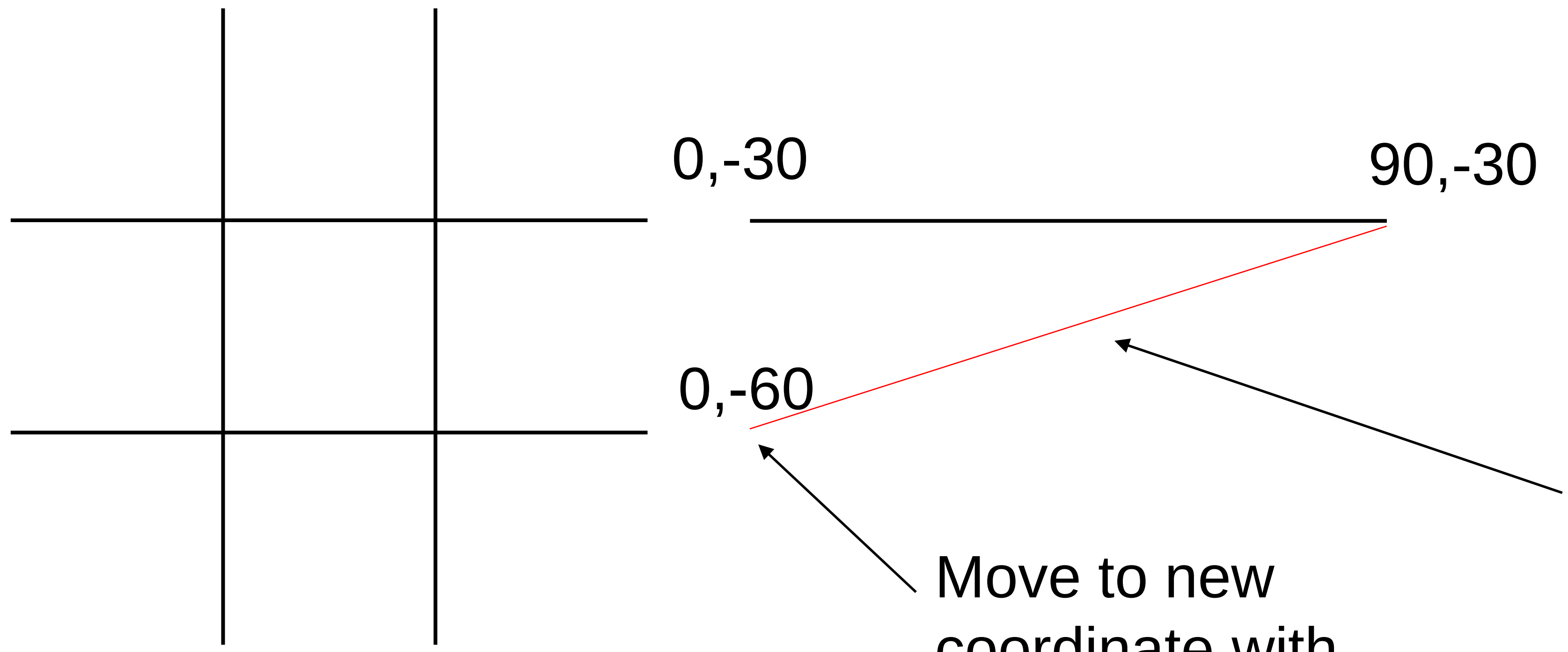
G0 X0.000000 Y-30.000000

S1000

G1 X90.000000 Y-30.000000

S0000 ← Pen up

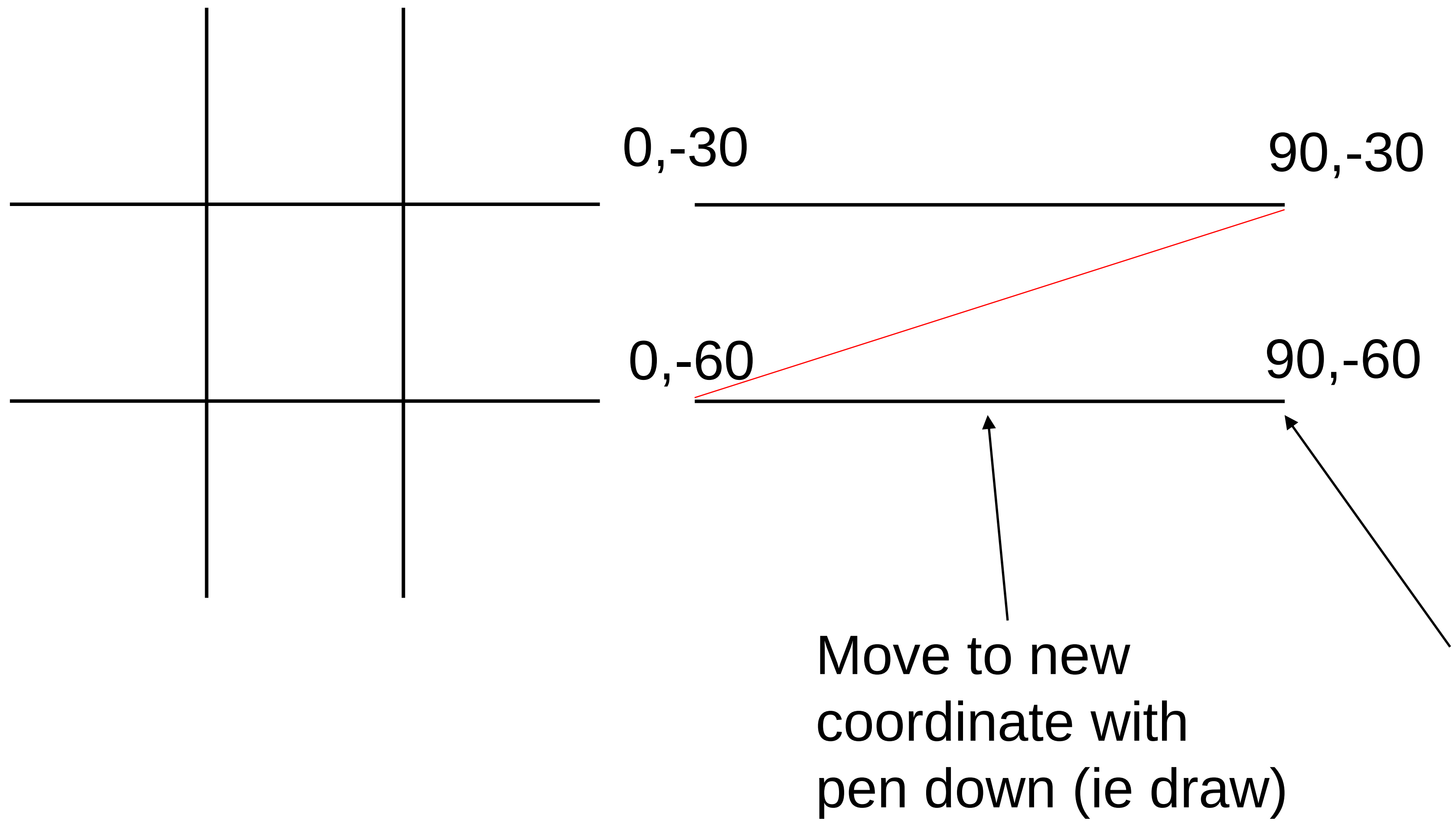
Drawing the grid – size 90mm (5)



Move to new coordinate with pen up (ie don't draw)

```
S0000
G0 X0.000000 Y-30.000000
S1000
G1 X90.000000 Y-30.000000
S0000
```
G0 X0.000000 Y-60.000000

Drawing the grid – size 90mm (6)



```
S0000
G0 X0.000000 Y-30.000000
S1000
G1 X90.000000 Y-30.000000
S0000
G0 X0.000000 Y-60.000000
```
S1000

G1 X90.000000 Y-60.000000

Fixed size 2D array and functions

A fixed size 2d array can be declared and initialised:

```
#define SIZE 3

int gridArray[SIZE][SIZE] = {{0,0,0},{0,0,0},{0,0,0}};
```

If passed to a function the size of at least the second dimension must be defined or the compiler can’t dereference the pointers:

```
void incrementArray( int array[SIZE][SIZE])
{
    int i,j;

    for (i = 0; i < SIZE; i++)
    {
        for ( j = 0; j < SIZE; j++)
        {
            array[i][j] = i+j;
        }
    }
}
```

GridArray.c

Function is called in usual way: `incrementArray( gridArray );`