SCC110: Software Development Course Work Programming Project.

Lancaster University

Project Title:	**Programming Project**
Moodle Submission Deadline:	**16:00 Friday, 1. June, 2018**
Assessment Mode:	**Demonstration via Wechat/Skype after 1. June, 2018** **Project report summarizing the design and implementation ideas submitted by 1. June**

Aims

In this final term of your first year, there is just a single programming task - a project that is designed to bring together many of the programming techniques that you have learned over the year. This project represents the final piece of practical coursework for SCC110.

You have a choice of which project to undertake, to be selected from the three choices below, and you need to pick just **one** of these projects. Read each of them carefully before choosing which project to undertake. Please note that the projects state which programming language you must use for that project (either C or Java).

Lancaster University

Assessment

This work will be assessed through a demonstration via wechat or skype and code inspection of your work. Alternatively, you could also submit a project report to summarize your project design and implementation details.

You will be asked to present your work in the demonstration. Be prepared to take the lead in demonstrating your project, and to answer questions about it posed by your markers.

- **You MUST submit your code to Moodle by the advertised deadline.**
- **You MUST demonstrate your work IN YOUR OWN LAB SESSION.**
- **The standard University regulation on permitting late submission of coursework DOES NOT APPLY TO THIS COURSEWORK ASSIGMENT.**

FAILURE TO ADHERE TO THE ABOVE WILL RESULT IN A MARK OF F4 BEING RECORDED.

Marking Scheme

Your work will be marked based on the following four categories. Your final grade will be determined based on a weighted mean of these grades according to the weighting shown in the table below.

Project Functionality - See individual project descriptions below for indicative levels of the functionality required.	**50%**
Code Structure and Elegance - Modularity of code - Use of appropriate data types and libraries - Use of appropriate language constructs (arrays, loops, functions, methods, classes)	**20%**
Code Style - Appropriate comments, code indentation - Appropriate name/scope of variables and functions / methods	**10%**
Project Presentation - Practical demonstration - Code review - Ability to answer questions	**10%**
Use of GIT version control - Clean and regular commits - Appropriate commit messages	**10%**

In all cases a grade descriptor (A, B, C, D, F) will be used to mark your work in each category. The following sections provide an indication of the level of functionality expected.

Markers can also recommend the award of a distinction (+) category overall if they feel a piece of work exhibits clearly demonstrable good programming practice. If you feel your work warrants a distinction category**, it is your responsibility to ensure that the marker is made aware of why**.

Lancaster University

Project Title: **Project 1: Centipede!**

Language: **C**

Project Overview:
Computer games are excellent way to stretch your programming ability, and practice both problem solving and developing your own solutions and algorithms. The aim of this project is to recreate an absolute arcade classic released by Atari in 1980 - Centipede!

Centipede (see Figure 1) is a vertically oriented shooter by Ed Logg and Dona Bailey. Designed intentionally to engage female players, the game consists of a centipede that winds it's way from top to bottom of the screen (where the player is located). The screen is populated with some number of mushrooms. When the centipede hits a mushroom, it drops one row towards the player and changes direction. The more mushrooms, the faster the centipede typically descends towards the player. The player can fire and must shoot the centipede. The shot segment becomes a mushroom. If the player shoots a segment other than the head, the centipede splits into two, effectively gaining a new head and both descend towards the player. Mushrooms can be destroyed, but take four shots to destroy. The arcade version of the game also features fleas, spiders and scorpions. See the Wikipedia description for more detailed discussion of the gameplay.

If you are unfamiliar with the game, watch the following youtube video demo: https://www.youtube.com/watch?v=dxoK8hosHjA, and read this overview: https://en.wikipedia.org/wiki/Centipede_(video_game)

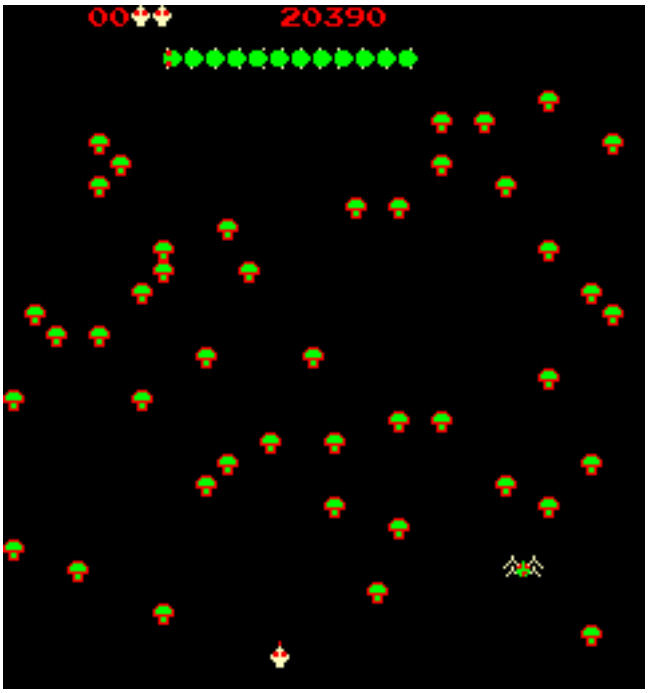


Figure 1: A Screenshot of Atari Centipede!

Lancaster University

Project Requirements:

Your task is to implement a working single player game of Centipede, based on the arcade original. A fully featured implementation of the game is a lot to ask in the time available, so the game can and should be simplified to make the problem more tractable. It's wise to start conservatively with the most basic functionality and add more features should the time be available. Minimally, you should aim to create a game that should:

- Display a single centipede and at least one player ship.
- Move the centipede back and forth descending one row at a time.
- Allow the player to move side to side.
- Allow the player to fire a projectile.
- Detect collisions between the projectile and the centipede.
- Keep score.

Advanced features (mushrooms, splitting when hit mid centipede, fleas etc.) can be added for more credit as time allows (see the marking scheme).

Getting Started

You'll need to do some research to learn the rules of operation of the game, and you'll need to learn how to use a library of C functions that can assist you in drawing ASCII based graphics to the screen – the ncurses library. To get stated, we recommend that you:

- Watch the YouTube video of the game to get an idea of how it works: https://www.youtube.com/watch?v=dxoK8hosHjA
- Make notes on the features of the game you can see – pay particular attention to the phases of gameplay (start, middle, end). What is displayed on the screen and how does it change?
- Read the introduction to the ncurses library (sections 1-1.3): http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/intro.html
- Look back over term 1 and remind yourself of how to log into 'UNIX' using the virtual machine and compile C programs.
- Try the 'hello world' example: http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/helloworld.html - you'll need to #include <curses.h> in your C file and compile using:

  `'gcc -o <target> <src>.c -lncurses'`

- Now try to create a C program that uses the ncurses library to let you control the position of a symbol (e.g. '*') on the screen using keyboard input. Tips: lookup mvprintw, getch and timeout in the man pages/online. See the example in the slides.

Marking Scheme:

Marks will be awarded according to the marking scheme shown at the start of this document. Marks for the 'Functionality' section will be awarded based on individual merit, but the following table gives an **indicative** overview of the level of functionality expected for each grade band:

A:	**Working program that meets the criteria for a B, plus:** - Richer game – e.g. other enemy types; multiple centipede head logic after split; levels (more than one wave of centipedes)
B:	**Working program that meets the criteria for a C, plus:** - Good basic centipede game, player takes a turn and can be destroyed (game can end) - Centipede can split forming multiple centipedes - Player can win by destroying all centipede pieces - Mushrooms
C:	**Working program that meets the criteria for a D, plus:** - Centipede descends following original logic - Player can fire - Projectile hits and destroys centipede (collision detection) - Score updated and displayed
D:	**Working program that:** - Basic single player game with a single centipede - Starts player and centipede, basic movement
F:	**No working program demonstrated, or program does not meet any requirements listed above.**

Lancaster University

Project Title: **Project 2: The Perilous Plank Puzzle Conundrum**
Language: **Java**

Project Overview:
Simulations are excellent ways to test your ability to implement programs to well-defined specifications… The aim of this project is to make use of the Java Swing APIs to create a simulation of the classical River Crossing game:

Figure 1: The Physical River Crossing Game

http://www.thinkfun.com/products/river-crossing

Project Requirements:

Your task is to implement a working, graphical puzzle game written in **Java** and based on the **Swing classes**. Your solution should allow the game to be played, and support multiple levels of play (as defined by the layout of pieces on the board – see gameplay section for details).

An example implementation of the game is shown below. **This is simply an example however – you are free to design your solution in any way you wish**. There is no requirement to build a solution exactly like this, with all the features shown (start with the basics and build from there):

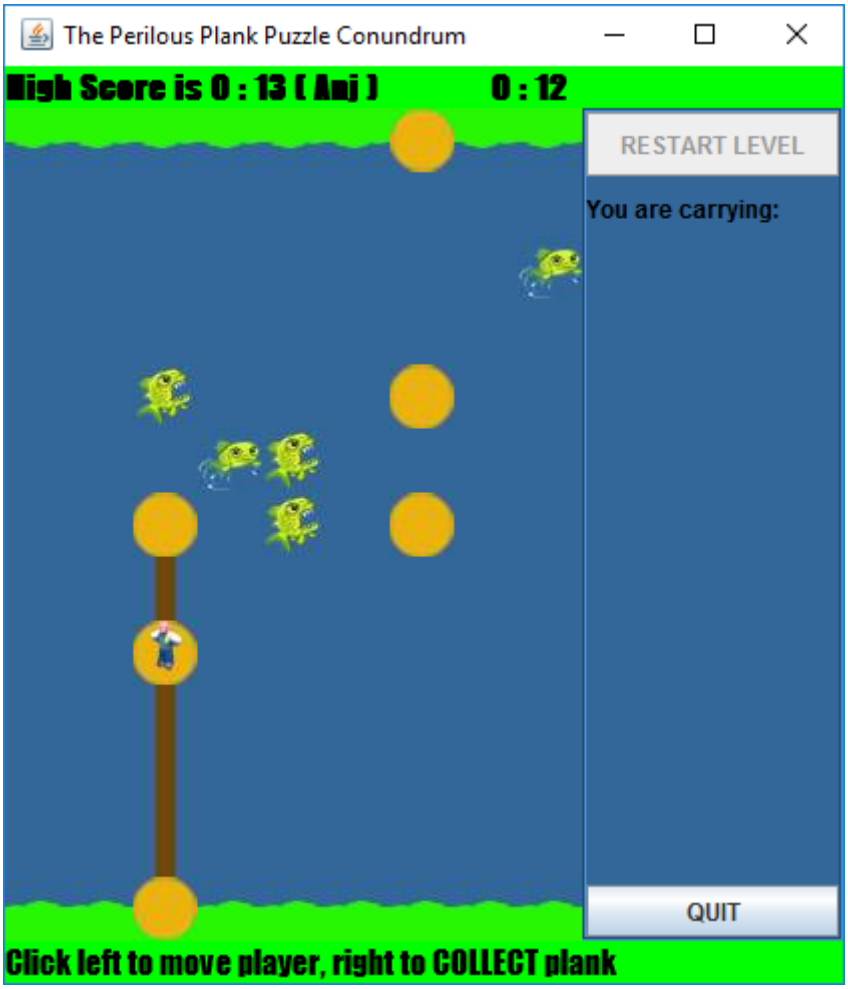


Figure 2: An Example River Crossing Application using Swing

River Crossing Gameplay:

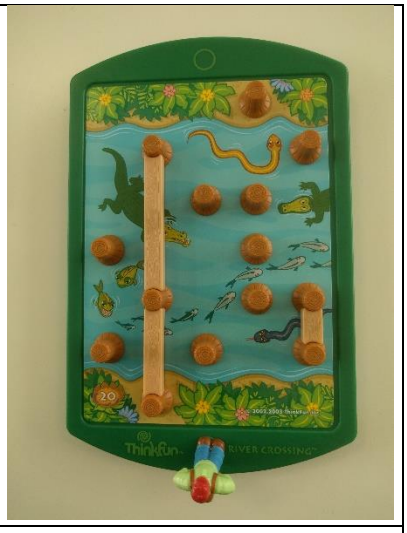
The principle of the puzzle is very simple. The game is played on a 9x13 grid of squares. Each position in the grid contains either a **stump**, a **plank** or **water**. The aim of the game is to move a player across the water from one side of the grid to the other. There are, of coure, rules however:

- The player must start from a designated start position. This will always be on a stump.
- The player completes a game level when she reaches a designated goal position. This will also always be on a stump.
- The player can move in any direction, but may only walk on stumps and planks.
- The player cannot swim, or walk on water.
- The player may pick up and carry a single plank at any point in time. A plank can be picked up if the player is standing on a stump directly connected to that plank.
- A plank may be placed between two stumps in either the horizontal or vertical direction, provided it is of the **exact** size to fit between those stumps.
- A plank may be one, two or three units long.
- A plank cannot be laid diagonally.
- A plank cannot cross or be laid on or across another plank.
- The player may rotate a plank to be either horizontal or vertical.
- The player may not carry more than one plank at a time.

There are 40 levels to the game, with varying levels of difficulty. Three examples are shown below. In all cases, the player starts at the stump at the bottom of the board, and completes the level by reaching the stump at the top.

Level 5

Level 20



Level 40

Figure 3: Game levels

Guidance:

- To simplify your task, we have provided a set of graphical images you can use in your program. You will find these on the SCC110 moodle page, alongside this specification. You are of course free to create your own graphics if you prefer, but we recommend you use these to reduce your workload.

- Consider the playing area of your user interface carefully. Given this is a 9x13 grid, that the player will likely want to click on, you might want to use a set of **JButton** instances to simplify this. Consider also how you might store these (117 individually named variables seems like a bad idea), and which Swing layout manager will help you.

- Note that Images in Swing applications are represented by a class called **ImageIcon**. The ImageIcon class contains a constructor that lets you create an instance of an ImageIcon from a given filename. In turn, the JButton class is able to create buttons showing images as well as text… In particular, the JButton class has a **constructor** that allows a JButton to be created from an ImageIcon. The following example shows an example of this in practice. Consider how this could be combined with the graphical images provided to create the core of you Swing User interface….

```
ImageIcon i = new ImageIcon("bank1.jpg");
JButton b = new JButton(i);
```

- Remember you can use an **ActionListener** to detect mouse clicks on a JButton, as we saw last term. If you want to detect other user input (such as right mouse button clicks or keyboard input), you may also want to read about the **MouseListener** and **KeyListener** interfaces too.

Marking Scheme:

Marks will be awarded according to the marking scheme shown at the start of this document. Marks for the 'Functionality' section will be awarded based on individual merit, but the following table gives an **indicative** overview of the level of functionality expected for each grade band:

A:	**Working program that meets the criteria for a B, plus:** - Multiple levels supported - Optimisation of player movement such that the player need not move just one square at a time. - The time taken to complete a level should be measured - A High Score should be maintained.
B:	**Working program that meets the criteria for a C, plus:** - The player can collect, move and place a plank - The player cannot collect a plank from, or place a plank at, an invalid location. - The player may rotate a planks orientation (to be either horizontal to vertical)
C:	**Working program that meets the criteria for a D, plus:** - The player is able to move around the board - The player cannot move to invalid locations (i.e. water)
D:	**Working program that:** - Shows a viable Graphical User Interface (GUI). - All stumps, planks and a player shown at a valid start location, consistent with one of the example levels given.
F:	**No working program demonstrated, or program does not meet any requirements listed above.**

Lancaster University

Project Title: **Project 3: Sensor Data Visualisation**
Language: **Either Java or C**

Project Overview:
Your task (should you choose to accept it) is to implement a program able to read data logged from wireless embedded systems recording environmental sensor data, and present it in a human-readable, high-fidelity way.

The data to read is from the supplied .csv files (which can be open in excel or a text editor if you want to see the contents), and this data should be presented as graphs (if you tackle this in Java) or via the console/command line (if you're tackling this in C).

Data Format

The data itself is in Comma Separated Value (CSV) format, with the following fields in this order:

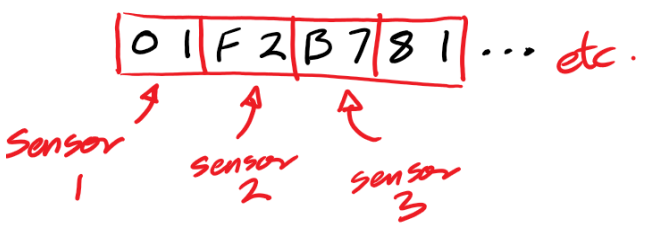
```
Time, Type, Version, Counter, Via, Address, Status, Sensor Data
```

Each field has the following meanings:

Field	Definition
Time	The time the sensor reading was sent, in *seconds from 1/1/2000 00:00:00*
Type	The type code to identify what device this is (should always be 0x20)
Version	What software version the device is running.
Counter	A rolling 8-bit, ever increasing number. Used to show how many messages are being *missed* by the receiver.
Via	Which receiver picked up this device's transmission.
Address	The address of the transmitter
Status	The status code of the device, as a bit-packed field. Any non-zero value should be presented as an error, but each bit in the field is a different error!
Sensor Data	10-bytes of sensor data, in hexadecimal. Each sensor is 1-byte long, so you should get 10 sensor values *per line.*

Apart from the 'Time' field, which is in decimal form, all of the fields are presented in hexadecimal number format.

The 'Sensor Data' field is special, as it contains many different sensor values one after the other in the following format:



Each of two-character pair is a separate sensor value, and is distinct from each of the others so they should all be read separately. An example of data in this format can be seen in full below, with the sensor field in the *last* column.

```
531113601,20,02,67,6,52000a57,00,15151316142172ffffff
531113620,20,02,93,6,52000a55,00,ffffff921a3effffffff
531113916,20,02,0c,6,52000a58,00,151bffffffffffffffff
531113905,20,02,dd,6,52000a57,00,1515131613166ffffffff
531113921,20,02,06,6,52000a55,00,ffffff911a3dffffffff
531114216,20,02,82,6,52000a58,00,151bffffffffffffffff
531114176,20,02,46,6,52000a57,00,1515131814166efffffff
531114221,20,02,79,6,52000a55,00,ffffff921a3dffffffff
```

An example chunk of one of the input files. Each line is newline '\n' separated, each column is comma ',' delimited

Because these records are from data sent over-the-air, they aren't all perfect! Some sensor data will have errors, and your program will be expected to gracefully decide what to do in these cases. Should the entire record be ignored? Should we ignore part of the record?

Hints and Tips for Java Implementations

If you want the user to select a file, you can use the built-in system file choosing dialogs with Java through the **JFileChooser** class. You can also restrict what files the user *can* open by using the **FileNameExtensionFilter** class.

```
// A Swing component for file request dialogs!
JFileChooser source = new JFileChooser();

// Use FileNameExtensionFilter to limit what files we want to get (by file extension)
FileNameExtensionFilter filter = new FileNameExtensionFilter("Comma Seperated Files", "csv");
source.setFileFilter(filter);

// Ask for a file, and check if the user actually selected one!
if( source.showOpenDialog( null ) == JFileChooser.APPROVE_OPTION ) {
    // Get the file the user selected
    File selectedFile = source.getSelectedFile();

    // Here is where you would kick off the reading/parsing stages
}
else {
    // Otherwise, complain that the user didn't choose a file!
    System.out.println( "No file chosen!" );
}
```

Basic usage for the JFileChooser class

See also:

- https://docs.oracle.com/javase/8/docs/api/javax/swing/JFileChooser.html
- https://docs.oracle.com/javase/8/docs/api/javax/swing/filechooser/FileNameExtensionFilter.html

Java does not have a pre-baked implementation of a graph for you to use, so you'll have to write your own class to handle how to draw a graph. The basic framework for doing this is shown below with random data points being shown on a grey graph area – yours should be much more feature rich!

```
import java.awt.*;

public class GraphComponent extends Canvas {
    public GraphComponent () {
        // Set your preferred size for the graph
        setPreferredSize( new Dimension( 300, 300 ) );

        // Make sure its visible!
        setVisible( true );
    }

    // Take over the 'paint' method, so we can draw our stuff!
    @Override
    public void paint( Graphics g ) {
        // Clear the entire canvas
        g.setColor( Color.WHITE );
        g.clearRect( 0, 0, getWidth(), getHeight() );

        // Draw a bit of background with a 10 pixel border
        g.setColor( Color.LIGHT_GRAY );
        g.fillRect( 10, 10, getWidth()-20, getHeight()-20 );

        // Draw some axis lines...
        g.setColor( Color.BLACK );
        g.drawLine( 20, 20, 20, getHeight()-20 );
        g.drawLine( 20, getHeight()-20, getWidth()-20, getHeight()-20 );

        // Draw some data! This is random, yours should be real :)
        for( int x=30; x<getWidth()-20; x+=10 ) {
            g.drawLine( x, getHeight()-20, x, getHeight()-20-
(int)(Math.random()*getHeight()*0.8) );
        }
    }
}
```

A simple random graph Swing component

The core of this is taking over the *paint* method, so we can do our own drawing, the rest of the detail in this example is just showing off some basic drawing functions.

See also:

- https://docs.oracle.com/javase/8/docs/api/java/awt/Canvas.html
- https://docs.oracle.com/javase/8/docs/api/java/awt/Color.html
- https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html

Hints and Tips for C Implementations

(Links here are to 'cplusplus.com' but are actually to the C-reference section of the site! Be careful you don't accidentally try and use a C++ function in C! It won't work!)

In C, as you'll be working with a lot of strings, it is worth getting familiar with <string.h> in the C standard library: https://www.tutorialspoint.com/c_standard_library/string_h.htm

Furthermore, as you're parsing CSV format input, you will likely want to pay particular attention to the following functions:

- strtok http://www.cplusplus.com/reference/cstring/strtok/
- sscanf http://www.cplusplus.com/reference/cstdio/sscanf/
- strtol and friends http://www.cplusplus.com/reference/cstdlib/
  (See the 'String Conversion' section)

To read strings in hex-format, you use the 'radix' field on the conversion functions to say the number is in hexadecimal format:

```
char * numberInHex = (char *)"123abc";

long longNumber = strtol(numberInHex, NULL, 16); // Radix is the LAST parameter
```

Converting a hex string to a LONG number.
Note that there are other functions for bytes, ints, etc.

All of the strto… functions work in a similar way.

Also, for your graph drawing, you can either do it all yourself, or use the ncurses library as described for Project 1 (or any other fancy drawing library you know of!)

Lancaster University

Project Requirements:

- Store the sensor data in a sensible data structure
- Provide ways the user can search for a device by address (and any other way you see fit!)
- Sort the sensors according to various categories (Time since last seen, number of records found, number of errors found, etc.)
- Flag up any errors in the data, and any cases where the devices report an error themselves (see the status field!)

For Java Programs

- Use Swing to present a high-fidelity interface for the data
- Draw graphs for historical data
- Must be able to view 'historical' data, not just the latest readings! (timelines, sensor-value-over-time line graphs, etc.)

For C Programs

- Present the user with a simple text-based interface (a menu or command interface)
- Show "graphs" using ASCII characters (perhaps use the ncurses library from Project 1 for this?)
- Allow the user to do multiple things per program run. The user should not have to restart the program each time!

Marking Scheme:

Additional features will be taken in to account to increase your mark, explore the dataset and work out ways that are useful to represent it. All the normal marks for code clarity, commenting and so forth also still apply 😊 I have included *a 400,000-record file*, which you can also read for a bonus.

A:	**Working program that meets the criteria for a B, plus:** - Can read the 100000-row data file and present some relevant output - Have a form of interactive user interface (must support multiple operations) - Min/Max values, average values - Basic feature detection (spikes in data) - Any extra features, such as exporting data from the program.
B:	**Working program that meets the criteria for a C, plus:** - Can read the 10000-row data file and present some relevant output - Filter the data for a particular field, such as the address/sensor ID - Have the data stored in an appropriate in-memory format
C:	**Working program that meets the criteria for a D, plus:** - Present historical data in some form of graph - Show some basic statistics on the data (error counts, sensor readings per device, etc.)
D:	**Working program that:** - Can read the 1000-row data file and present some relevant output
F:	**No working program demonstrated, or program does not meet any requirements listed above.**