University of Leeds **School of Computing**
Procedural Programming COMP1711/XJCO1711
Semester 1, 2020-2021

Coursework 3

100 Marks (40% of the total module mark)

To be submitted before: 23:00 (UK time) on 11 December 2020
Late penalties: 5% will be deducted from the overall mark of this coursework for every late day

Skills Tested

This coursework will test your ability to write C code that correctly uses structures, character strings, pointers, and file i/o. You will also learn how a program can be divided across multiple source files.

The Brief

You will write a program that allows people to name pairs of stars in a hypothetical universe by their names.

The Details

An astronomer wanted to surprise their spouse on their anniversary by showing them a simulation of the Big Bang on a computer, then naming the closest pair of stars created by this Big Bang with their names. Big Bang is the name of a cosmological theory that assumes that the universe started from the expansion of a single very high-density singularity.

Fortunately, the program is not supposed to go anywhere close to a real simulation of the Big Bang. For the purposes of this simple program, our Big Bang simply fills a flat rectangular space with stars scattered at random positions. The program then allows a user to find the closest pair of unnamed stars in this universe and assign the user's name to the first star in this pair, and the name of the user's spouse to the second star.

The Task

Write a command driven program that accepts the following commands:

The bang command

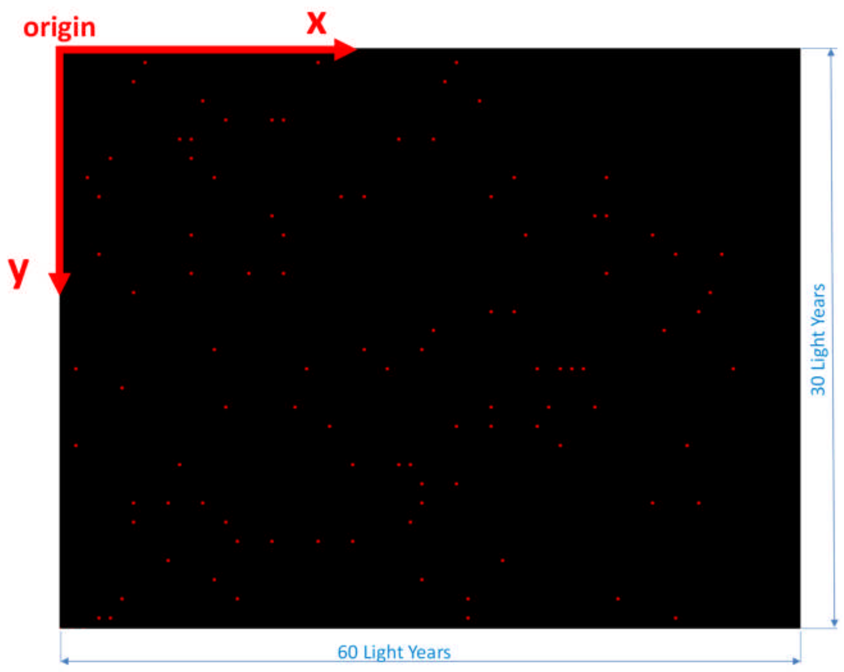
This command creates a hypothetical universe comprised of stars only. The stars should be randomly scattered across the universe.

We are assuming that the universe is flat and rectangular and is 60 light years long and 30 light years wide. A light year is a unit of measure of distance and is equal to the distance that light can travel in one year (5.879x10^{12} miles). Do not get daunted by this jargon, just assume that your space is a rectangle 60 units long and 30 units wide.

The bang command takes one integer argument representing the number of stars to be created. For example, if the player types:

```
>>> bang 100
```

the program will create 100 stars randomly distributed within the universe (the 60x30 rectangle). The position of a star is determined by two coordinates, x and y. No two stars should have the same coordinates. For simplicity, assume that x and y are both integers. Also assume that the origin of the coordinate system is at the upper left corner of the universe, with the x axis pointing to the right, and the y axis pointing downwards, as shown in the following figure:



Note that in the above figure, the aspect ratio (ratio of height to width) of the universe is distorted. The reason for this will be explained later on.

It is important to keep the origin of the coordinate system at the upper left corner, and the directions of the axes as shown. Changing these assumptions could make it more difficult to implement the drawing functions of the game.

Each star must have a unique serial number (id) generated by the program when the universe is created. Initially, stars have no names, but players can later name stars using the *name* command (see below).

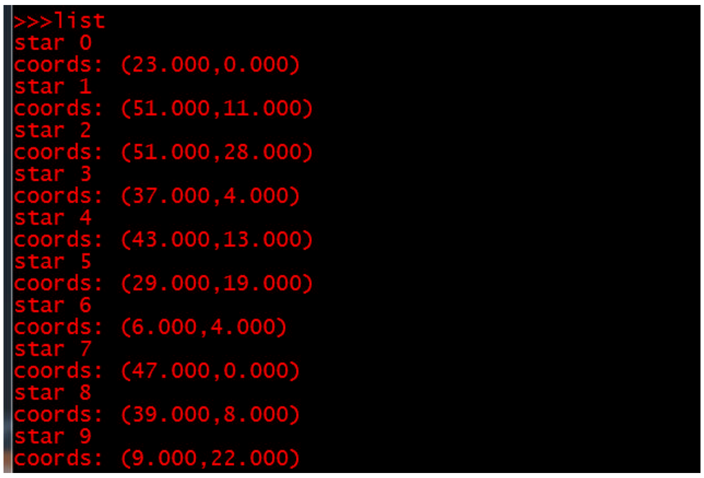
If the bang command is issued again (after a universe has been created), the existing universe is destroyed and a new one is created in its place.

The *list* command

This command simply prints a list of all stars in the universe. For each star, the command prints the star's serial number, name (if the star is named), and the star's x and y coordinates. For example, if the player types:

```
>>> list
```

the program prints a list similar to this one:

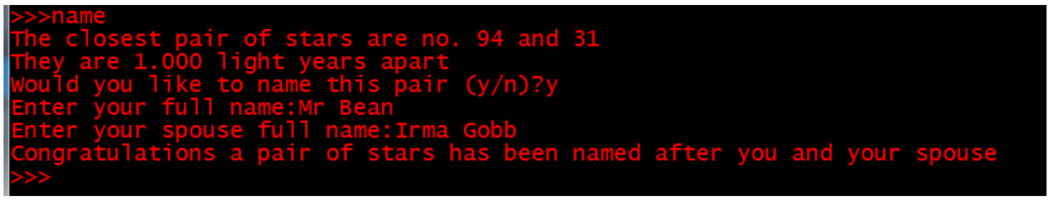


```
>>>list
star 0
coords: (23.000,0.000)
star 1
coords: (51.000,11.000)
star 2
coords: (51.000,28.000)
star 3
coords: (37.000,4.000)
star 4
coords: (43.000,13.000)
star 5
coords: (29.000,19.000)
star 6
coords: (6.000,4.000)
star 7
coords: (47.000,0.000)
star 8
coords: (39.000,8.000)
star 9
coords: (9.000,22.000)
```

Note that in the above list none of the stars has a name, which is the situation when a new universe is created.

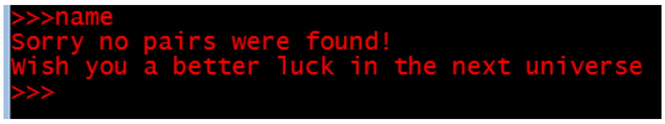
The *name* command

This command is used to find the closest pair of stars that has **not** been named yet and allow the user to name this pair. The program prompts the user to enter their name and that of their spouse. The first star in the pair is named after the player, and the second is named after the player's spouse. Once a pair of stars is named, this pair is permanently reserved for its 'owners' and cannot be renamed by other players. Here is an example of the name command:



```
>>>name
The closest pair of stars are no. 94 and 31
They are 1.000 light years apart
Would you like to name this pair (y/n)?y
Enter your full name:Mr Bean
Enter your spouse full name:Irma Gobb
Congratulations a pair of stars has been named after you and your spouse
>>>
```

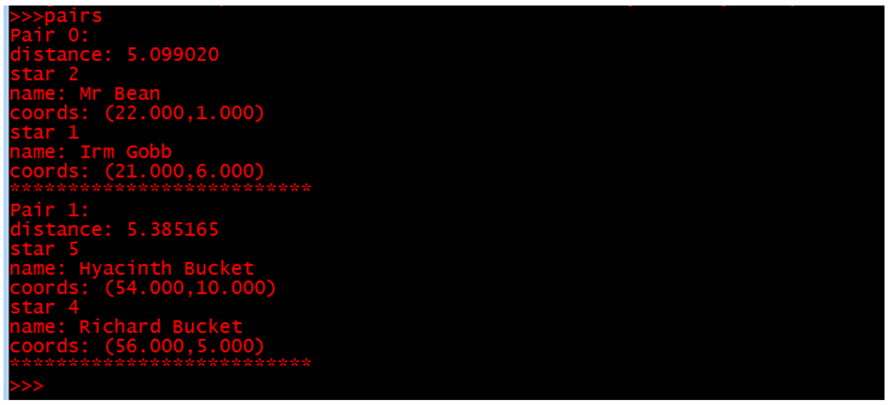
However, if all pairs have already been named, the program prints a message similar to this:



```
>>>name
Sorry no pairs were found!
Wish you a better luck in the next universe
>>>
```

The *pairs* command

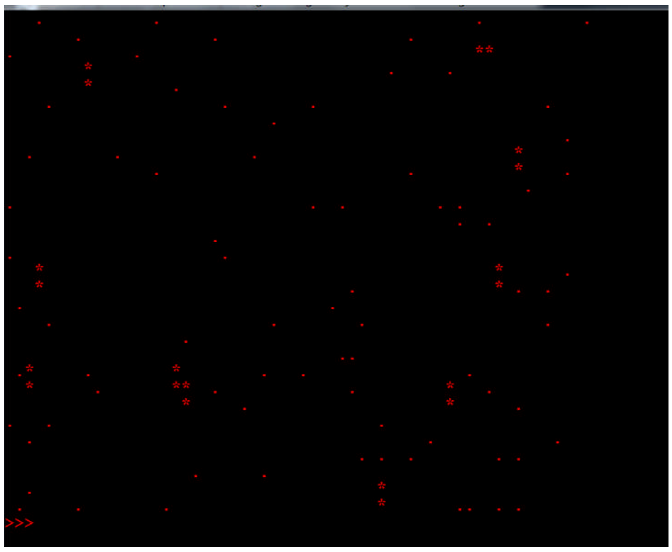
This command prints a list of all star pairs that have been named so far. For each pair, the program prints the pair's number, the distance between the two stars of this pair, and the details of the two stars. Here is an example:



```
>>>pairs
Pair 0:
distance: 5.099020
star 2
name: Mr Bean
coords: (22.000,1.000)
star 1
name: Irm Gobb
coords: (21.000,6.000)
*******************************
Pair 1:
distance: 5.385165
star 5
name: Hyacinth Bucket
coords: (54.000,10.000)
star 4
name: Richard Bucket
coords: (56.000,5.000)
*******************************
>>>
```

The *draw* command

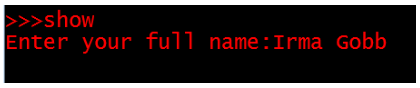
This command is used to draw the universe. Named stars appear as asterisks (*****), while unnamed stars appear as dots (**.**). Here is an example:



You will **NOT** be using any graphics library to draw the universe. Instead, you will use a simple trick to convert the standard terminal to a primitive drawing window. This will be explained below.

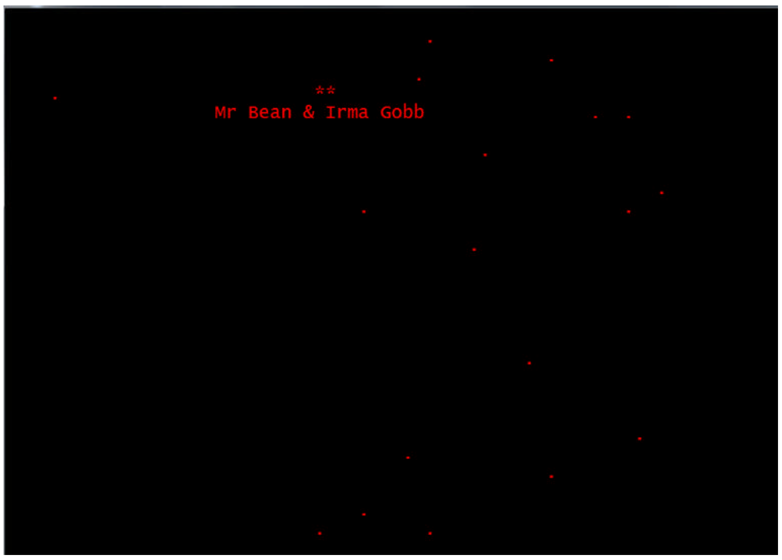
The *show* command

This command is used to display the names of the couple who own a pair of stars. When this command is executed the program prompts the user to enter their name as shown below



```
>>>show
Enter your full name:Irma Gobb
```

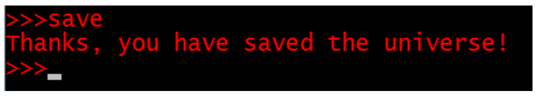
The program then searches for a pair of stars named after this user, and if a pair is found, the program displays the names of the couple who own the pair under the stars of this pair, as shown in the following example:



```
                  **
       Mr Bean & Irma Gobb
```

The *save* command

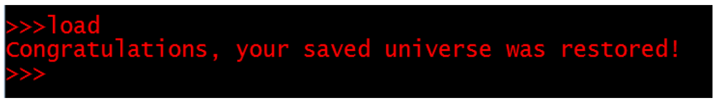
This command is used to save the universe. By this we don't mean saving the universe from the evils of a supervillain. Instead, this command saves the program's data into a file. The command should save all stars and all named pairs in the universe into a **binary** file called universe.bin located in the program's directory. Here is an example:



```
>>>save
Thanks, you have saved the universe!
>>>_
```

The *load* command

This command is used to load (read) saved data from the universe.bin file. When the command is executed the program reverts to the point at which it was saved.



```
>>>load
Congratulations, your saved universe was restored!
>>>
```

The *quit* command

This command is used to terminate the program.

Implementation Guidelines

<u>Generating random numbers</u>

To generate random values for star coordinates, you can use the *rand* function. This function is defined in the <stdlib.h> library. When the function is called it returns a random integer between 0 and RAND_MAX. The value of RAND_MAX is system dependent but is guaranteed to be at least 32767. You can scale the random number returned by *rand* to any required range [0, x-1] by applying the modulo operator on the value returned by the function, like this:

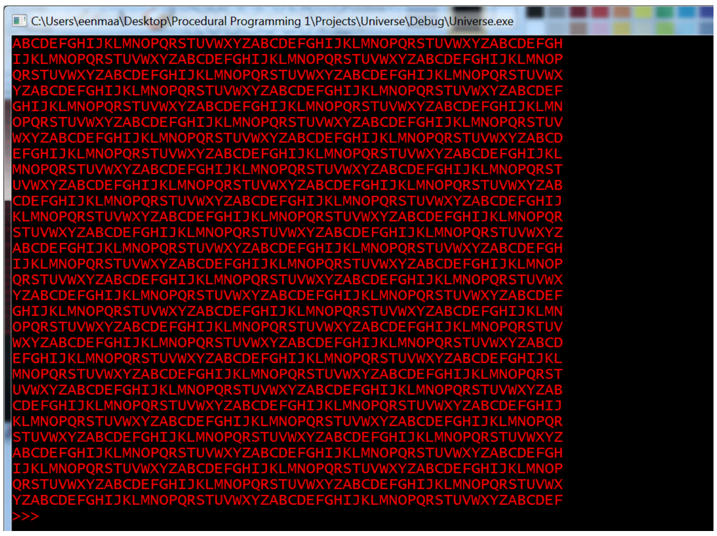
```
int r = rand() % x;                    // r is a random number in the range [0, x-1];
```

Unfortunately, the *rand* function is a *pseudorandom* number generator, which means that it will always generate the same sequence of random numbers every time you run the program. To get a different sequence each time, you can use the *srand* function (also defined in <stdlib.h>). The srand function 'seeds' the random number generator with an initial value. By changing the initial value, we can get different random number sequences. We need a value that is different every time we run the program. This can be some distinctive runtime value, like the value returned by the *time* function defined in the <time.h>. The time function returns the time in seconds that elapsed since some past point in time. You should seed your random number generator only once at the beginning of your program like this:

```
time_t t;
srand((unsigned) time(&t));
```

<u>Using the terminal as a graphics window</u>

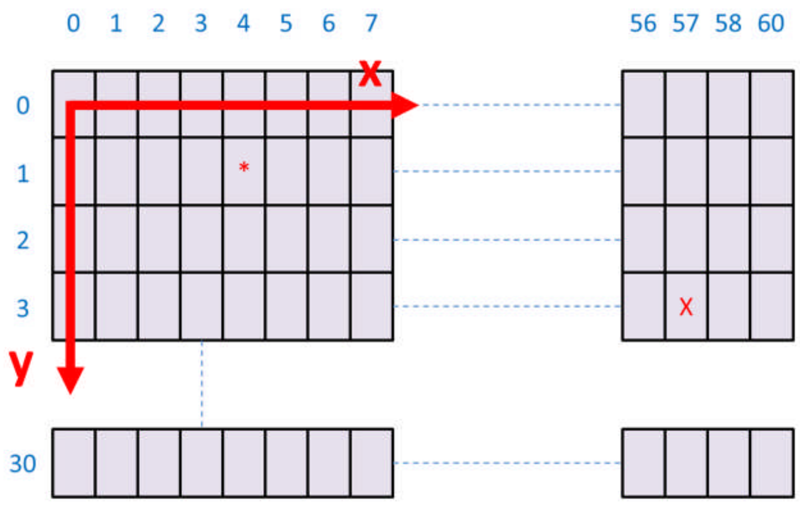
To 'draw' the stars of the universe on the terminal, we will use a simple trick based on the fact that the terminal prints characters in a rectangular grid of rows and columns. The cursor that determines the position of the next character to be printed is initially at the top left corner of the terminal window. When a single character is printed, the cursor moves exactly one column to the right. When the new line character ('\n') is printed the cursor moves to the first column in the next row. The following figure shows how the terminal will look like when 30 rows of characters, with 60 characters (columns) per row, are printed.


C:\Users\eenmaa\Desktop\Procedural Programming 1\Projects\Universe\Debug\Universe.exe

```
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGH
IJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP
QRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWX
YZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF
GHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
OPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUV
WXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCD
EFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKL
MNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
UVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZAB
CDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJ
KLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQR
STUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGH
IJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP
QRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWX
YZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF
GHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
OPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUV
WXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCD
EFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKL
MNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
UVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZAB
CDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJ
KLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQR
STUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGH
IJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP
QRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWX
YZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEF
>>>
```

In this sense, the terminal can be considered as a very coarse grid of 'pixels', just like any other graphics display. However, the elements of the grid are characters rather than pixels. Notice that the above grid appears rather squarish although it has twice as many columns as it has rows. This is because the height of the font used to display the characters is almost twice as its width. Most Latin fonts are like this, but you may find one on your machine that is not.

To draw things on the terminal, you will need a two-dimensional array (matrix) of characters. We will call this matrix the *frame buffer* because it will be used to prepare the frame (picture) of the universe before printing it on the screen. Each row in this matrix corresponds to one row of characters on the terminal. **The frame buffer will have 31 rows and 61 columns**, as shown below:



The Frame Buffer Matrix

Now, to draw a star whose (x, y) coordinates are (4, 1) for example, we simply store the asterisk (*) in row 1 and column 4 of the frame buffer as shown above. And, to 'draw' the letter X in location (57, 3) we put 'X' in row 3 and column 57 of the frame buffer.

After all the stars of the universe are stored in their correct positions in the frame buffer, the whole frame buffer can be displayed on the terminal in one go.

Submission Instructions

- Write the program in standard C. If you write your code in any other language, it will NOT be marked, and you will get a zero for this coursework.
- Gradescope will be used to mark this coursework. Detailed instructions on how to prepare and submit your code to Gradescope will be published on Minerva.
- Four files - called uni.h, graph.c, logic.c, and main.c - are provided for you to start developing your code. Do not alter anything in uni.h and do not submit it to Gradescope.
- File uni.h contains some function prototypes. You must implement the body of these functions in the graph.c, logic.c and main.c files. You will also need to use these functions to implement the program functionality.
- Make sure that you fill in your name, University number, and the date you started working on the assignment in the space provided at the top of each file.
- This is an individual project, and you are not supposed to work in groups or pairs with other students.
- **Please be aware that plagiarism in your code will earn you a zero mark and will have very serious consequences. It is much better to submit your own partially finished work, than to fall into the trap of plagiarism. We use software to detect plagiarism, so if you do work with someone else, or submit someone else's work it WILL be detected.**

Finally, let's hope that the astronomer's wife will be impressed by the game after all this hard work.

Marking Scheme

Graphics Functions

Function clearBuffer ()	(4 marks)
Function plot ()	(2 marks)
Function peek ()	(2 marks)
Function writeAt ()	(6 marks)
Function showBuffer ()	(6 marks)
Program Logic Functions	
Function findStarByXY ()	(5 marks)
Function bigBang ()	(12 marks)
Function pointDistance ()	(2 marks)
Function closestPair ()	(12 marks)
Function nameStar ()	(5 marks)
Function findPairByName ()	(4 marks)
Function saveUniverse ()	(6 marks)
Function loadUniverse ()	(6 marks)
Interface Functions and the User Interface	
Function printStar ()	(2 marks)
Function listStars ()	(2 marks)
Function listPairs ()	(4 marks)
Function drawUniverse ()	(6 marks)
Function tagPair ()	(6 marks)
The user interface works correctly	(8 marks)

Total mark:	**[100 marks]**