University of Leeds **School of Computing**
Procedural Programming COMP1711
Semester 1, 2017-2018

Individual Mini-Project 3

(Covers Assignments 6 and 7)

40 Marks

Submission Deadline: 08/12/2017 at 10:00 am

The main objectives of this project is to help you learn how to use structures, character strings, and file i/o.

This project is comprised of two tiers. Tier 1 is compulsory and must be completed by all students. If you satisfy all the requirements of Tier 1, you can earn the full marks for the project.

Tier 2 is optional, and you can choose to do it if you would like to stand out from the crowd. Tier 2 is not marked, but if you accomplish it correctly, your name will be written in the Distinction List on the Minerva page of the procedural programming module, and if your code is exemplary it will also be placed on Minerva for all your colleagues to see.

Tier 1 (Compulsory)

The Brief

You will write a program for a simple game that allows couples to name pairs of stars in a hypothetical universe with their names.

The Details

The Problem

An astronomer wants to surprise his wife on their anniversary by showing her a simplistic demonstration of the Big Bang on his computer, then name 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*.

Førtunately, the program is not supposed to go anywhere close to a real simulation of the Big Bang. For the purpose of this simple game, our Big Bang simply fills a flat rectangular space with stars scattered at random positions. The program then allows a player to find the closest pair of stars in this hypothetical universe and name this pair with the name of the player and that of his or her partner.

The Task

Write a command driven game that accepts the following commands:

The *bang* command

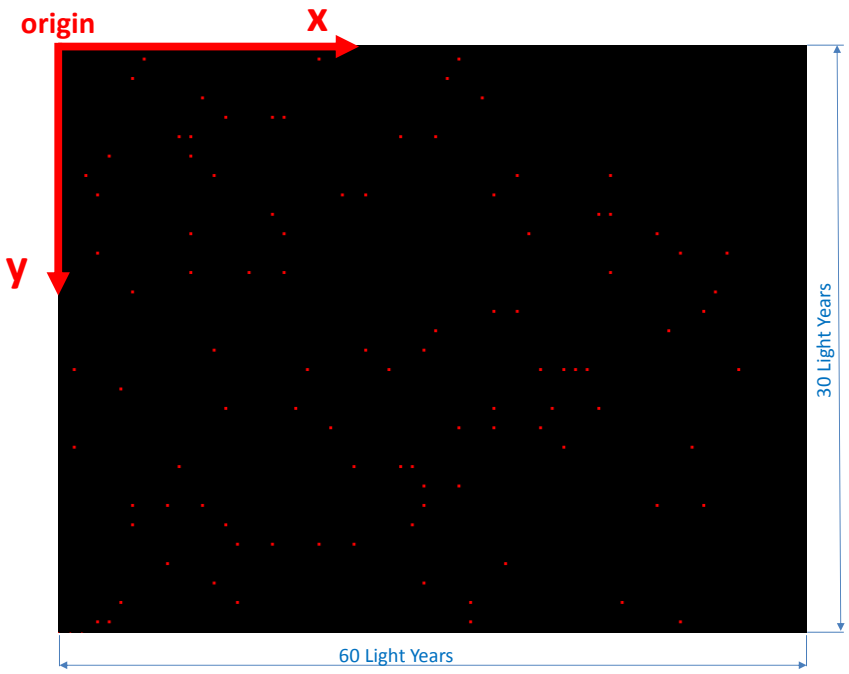
This command creates a hypothetical universe comprised of stars only. The stars should be randomly scattered across the universe.

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 (5.879×10^{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 creates 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 commands of the game.

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

If the bang command is executed 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 the 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:

```
>>>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 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 player to name this pair. The program prompts the player to enter his name and that of his or her spouse. The first star in the pair is named after the player, and the second is named after his or her spouse. Once a pair of stars has been named, the 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
>>>
```

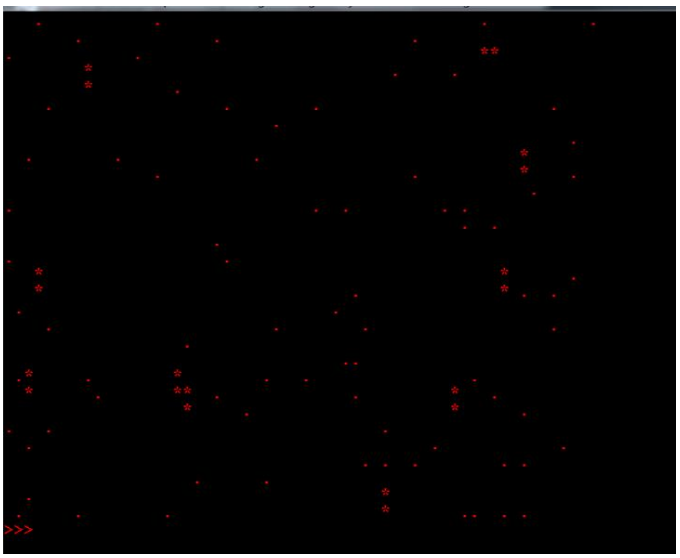
The *pairs* command

This command prints the list of all pairs of stars 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 Linux (or Windows) terminal to a very primitive drawing window. This will be explained later on.

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 player to enter his or her name as shown below

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

The program then searches for a pair of stars named after this player, 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:

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, it is used to save all the game's data into the file system. 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 current 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 game 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 Instructions

The data structures

In this project, you will need to define two types of structures (i.e. structs), one for stars, and one for named star pairs. The star structure contains all the data about a single star. This should include:

- The star serial number (an integer).
- The star name (a character string).

- The star x and y coordinates (doubles).

You may also decide to add other data members to the star structure.

The pair structure holds data about a named pair of stars. It should have the following data members:

- The serial number of the first star in the pair (an integer).
- The serial number of the second star in the pair (an integer).
- The distance (in light years) between the two stars (a double).

You may also decide to add some other data members to this structure.

You will need to store the game data in two arrays. The first is an array of all stars in the universe, and the latter is an array of all named pairs.

Generating random numbers

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 x operation on the value returned by the function, like this:

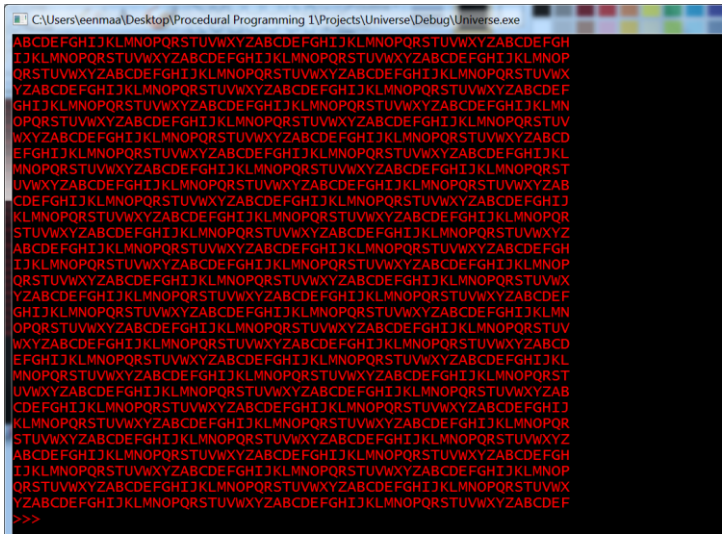
```
int r = rand() % x;                    // r is a random number in the range [0, x-1];
```

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, 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));
```

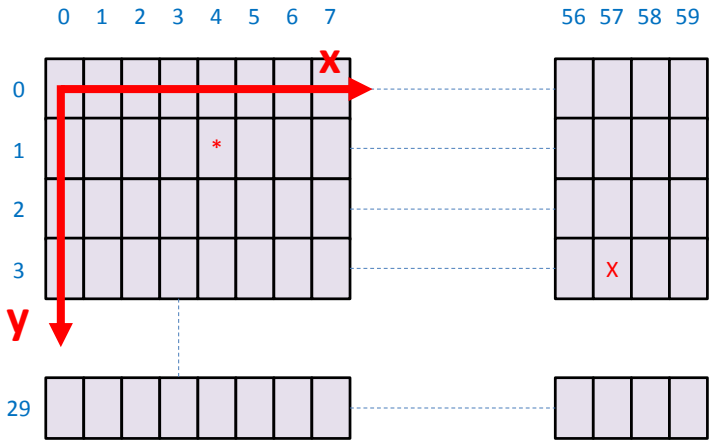
Using the terminal as a graphics window

To 'draw' the stars of the universe on the terminal, you 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 exactly 60 characters (columns) per row, are printed.



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 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 30 rows and 60 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 is printed on the terminal in one go.

You will write a number of functions that facilitate the above graphics operations. These are:

```
void Plot (int x, int y, char c); // store c in row y and column x of the frame buffer
void Clear ();   // clear (erase) the whole frame buffer by filling it with white space (ASCII 32)
void WriteAt (int x, int y, str[]) // copy the string str to the frame buffer starting at position x,y
void Refresh ()   //clear the terminal and print the entire frame buffer
```

To clear the terminal you can call the Linux clear command from within your C program, using the *system* function, like this:

```
system ("clear");  // this will clear the terminal and send the cursor to the top left of the screen
```

If you are using a Windows based system, use this: system ("cls");

General Guidelines

- In this project you can use any C language statement or construct that you have learned in this course.
- Write the program in standard C. If you write your code in any other language, it will NOT be assessed and you will get a zero mark.
- This is an individual project, and you are not supposed to work in groups or pairs with other students.
- 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.

To simplify program development, you can divide your work into individual Work Packages (WP) as follows:

WP1: implement and test the CLI. At this stage of program development, the CLI can read and distinguish various commands but cannot execute any of them.

WP2: implement and test the graphics interface functions (*Clear, Plot, WriteAt,* and *Refresh*)

WP3: Implement the various instructions, one at a time, in the order they have been described above. Test each instruction as you proceed.

WP4: Test and debug your entire program.

WP5 (optional): Implement the second part of the game (see below).

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

Marking Scheme

The CLI works correctly	(3 marks)
The 4 graphics functions are implemented correctly and efficiently	(8 marks, 2 each)
All 8 commands of the game work correctly and efficiently	(24 marks, 3 each)
(The Quit command has no marks)	
The program is well structured, clear, and efficient	(4 marks)
The program uses comments properly	(1 mark)

Tier 2 (Optional)

We would like to take this game a little bit further by assuming that the universe is comprised of stars and planets. The *bang* command now has two arguments representing the number of stars and planets respectively.

We will also assume that stars die after a certain average age, for example 1 billion years. When a star dies it shoots out of the universe in a random direction (like a shooting star). **We know that real stars do not do this when they die and that shooting stars are not even stars**, but we will accept this piece of fake science to complete the plot of the game. If the falling star passes within a certain distance from a planet, that planet is destroyed.

The idea of this part of the game is to allow a player to examine the universe just after the Big Bang and try to find a planet that the player thinks is more safe from shooting stars than others, and make a bet on how long the planet will survive before being hit by a falling star.

The player can then run a simulation and watch the events that unfolds within the period of time he made a bet on. If the period elapses before the player's planet is destroyed, the player wins.