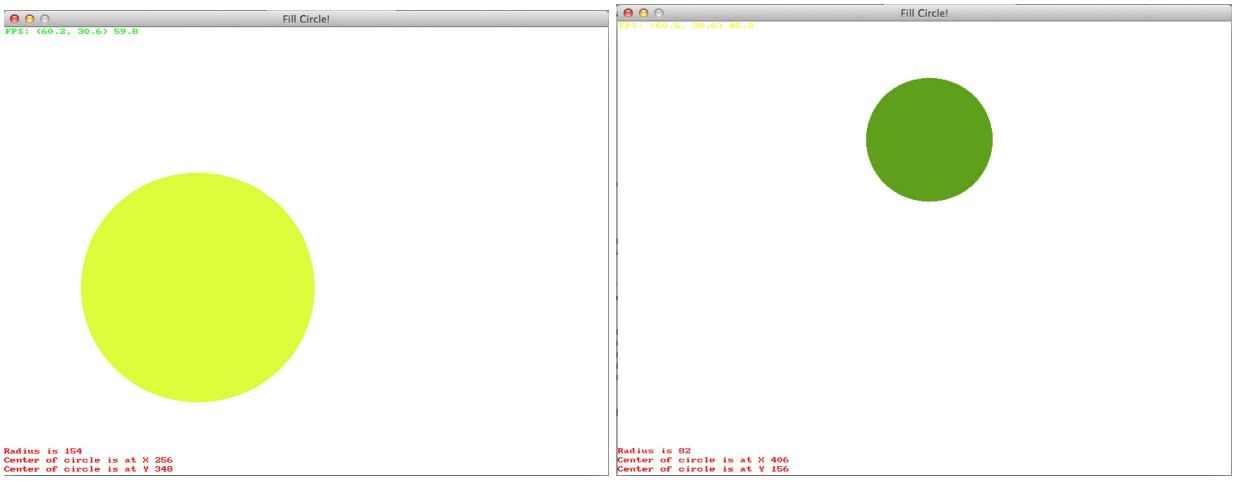
Introduction to Programming Glossary

By Reuben Wilson (9988289)

Programming Overview

The implementation of control flow makes programs so much more interactive and dynamic. Before being introduced to control flow this semester, all the code that we had written was great in terms of execution and the desired output, but the programs were extremely limited with what they could do. They would only perform tasks in the order in which they were specified, and once the desired output was met, the program would terminate. This seems robotic and impractical because the user can only interact with the program within its specified bounds.

Using control flow in programs drastically improves useability by implementing a system, which makes programs more dynamic and interactive. How is this achieved? By using control flow, you can alter the sequence of execution of code within the program based on conditions, which may or may not be met. For example, by using control flow, I can draw a circle to the screen; change its size, colour and location, amongst other variables; based entirely on the user's input. Below are some nifty screenshots of a program I made, which does just that:



For a greater understanding of how I implemented control flow in the program, which draws these circles, refer to the code provided at the back.

Programming Concepts:

The following concepts are central to procedural programming.

Concept	Description

Control Flow	Control flow is an excellent way for programmers to implement structured control in their programs based on special dependencies. For example, control flow can be used to execute a section of code only when requirements for that code have been met. Refer to the snippet of code below:
	`// If the 'down' key is pressed then add '3' to the value of 'circleY'`
	`// This will move the circle on the screen down 3 pixels`
	`if KeyDown(vk_Down) then`
	`  circleY += 3;`
	So, from the above example, the instructions are only executed when the user presses the *down arrow* on the keyboard.

Selection	Remember that a sequence of code represents the specific order in which the code is executed. By using selection, you can alter which branch of code in the sequence is executed. This can be achieved by implementing Boolean values (true or false) or while statements.

Repetition	Is used to repeat a section of code in two ways. They are *the pre-test loop,* which will only repeat a section of code if a condition is met and *the post-test* loop, which will repeat a section of code at least once before checking any conditions.
	The short snippet of code below is an example of a *pre-test loop*:
	`i := 0;`
	`while (i < 100000) do`
	`begin`
	`  Write(' ashamed');`
	`  i := i + 1;`
	`end;`
	You can see the value of variable *i* is assigned the value of *0* to begin with, the loop then checks the value of *i,* and as long as it is less than 100000, the loop will repeat.
	
	The short snippet of code below is an example of a *post-test loop*:
	`Repeat`
	`  WriteLn('Please select an option, ',name, '.');`
	`  WriteLn(' - [S]illy name');`
	`  WriteLn(' - [H]ello world');`
	`  WriteLn(' - [Q]uit');`
	`  WriteLn('Option -> ');`
	`  ReadLn(option);`
	
	`  case option of`
	`  's', 'S': SuperSillyName(name);`
	`  'h', 'H': WriteLn('Hello World!');`
	`  'q', 'Q': WriteLn('Peace...');`
	`  else`
	`  WriteLn('Shiieeeet ',name, '. Please try again.');`
	`  end;`
	`  WriteLn();`
	`until (option =  'q') or (option = 'Q');`
	So you can see here that the repeat loop is executed before any conditions are checked.

Indentation	Correct indentation is extremely important because It makes code easy to read and follow. If something is wrong, debugging is much easier with correct indentation because you can follow each section of codes sequences step by step.

Structured Programming

Sequence:

A *sequence* is a specific list of instructions, which can be made up of any number of events. When the program is run, none of the listed instructions can be skipped. Each has to be run in the order in which they are specified.

E.g. This particular sequence of codes states a specific order in which the procedures are run. The order is as follows:

1. *OpenGraphicsWindow();*
2. *LoadDefaultColors();*
3. *LoadResources();*
4. *DrawDoor();* etc.

The program relies on this sequence to be executed in order to run properly.

```
OpenGraphicsWindow('Knock Knock...', 800, 600);

// Call the LoadDefaultColors procedure -> effect is SwinGame colors are initialised
LoadDefaultColors();

// Call ShowSwinGameSplashScreen -> effect is SwinGame splash is shown
ShowSwinGameSplashScreen();

// Call LoadResources -> effect is our images/sounds are loaded into SwinGame for use
LoadResources();

// Call DrawDoor -> effect is the background door is shown... with delay for use to see it.
DrawDoor();

// Call ShowKnockKnock -> shows knock knock text and plays sound
ShowKnockKnock();

// Call CloseAudio -> turns off SwinGame's audio
CloseAudio();

// Call ReleaseAllResources -> Release all of the things we loaded...
ReleaseAllResources();
```

Selection:

By using selection, you're not bound by a set sequence of instructions. Selection enables you to select sections of code to be run, providing certain circumstances are met, meaning the same section of code can be run as many or as little times as needed for the functionality of the program. In relation to what I've covered in Pascal so far, refer to the snippet of code below for an understanding of how I've implemented selection using *if* statements:

```
// If the 'r' key is pressed, assign the 'circleX' & 'circleY' variables random values between screen width and height
// This will draw the circle in a random location on the screen
if KeyTyped(vk_r) then
    circleX := Rnd(ScreenWidth());
if KeyTyped(vk_r) then
    circleY := Rnd(ScreenHeight());
// If the 'c' key is pressed, assign 'clr' a random colour.
// This will change the colour of the circle to a random colour
if KeyTyped(vk_c) then
    clr := RandomColor;
// If the left mouse button is pressed and the mouse pointer is inside the circle,
// assign 'myCircleIsFilled' with the boolean value that it is not.
// E.g. If it is 'True', assign it 'False'. If it is 'False', assign it 'True'
// So, only if the mouse is clicked inside the circle, the circle will become solid, filled with colour
if MouseClicked(LeftButton) and PointInCircle(MouseX(), MouseY(), circleX, circleY, radius) then
    myCircleIsFilled := not myCircleIsFilled;
// If the 'left' key is pressed then subtract '3' from the value of 'circleX'
// This will move the circle on the screen 3 pixels to the left
if KeyDown(vk_Left) then
    circleX -= 3;
// If the 'right' key is pressed then add '3' to the value of 'circleX'
// This will move the circle on the screen 3 pixels to the right
if KeyDown(vk_Right) then
    circleX += 3;
```

Repetition:

By using repetition, you can repeat any section of code as many or as little times as you like depending on any dependencies that may or may not have to be met. For example, an excellent scenario of where repetition would be useful is in a small computer game, you want the code that makes the game functional to be repeated for the duration of the game, or until the user closes the application. In regards to how I've implemented repetition in Pascal, refer to the snippet of code below, which uses a *repeat* statement to execute a block of code until the user closes the application:

```
// create a loop which repeats a 'while' statement until the user closes the program
// So, while 'drawCount < 1000', draw 1000 triangles to the screen with random colour and sizes.
    repeat
        ProcessEvents();
            while (drawCount < 1000) do
                begin
                    triangleX1 := Rnd(ScreenWidth());
                    triangleY1 := Rnd(ScreenHeight());
                    triangleX2 := Rnd(ScreenWidth());
                    triangleY2 := Rnd(ScreenHeight());
                    triangleX3 := Rnd(ScreenWidth());
                    triangleY3 := Rnd(ScreenHeight());
                    triangleColour := RandomColor;
                    DrawTriangle(triangleColour, triangleX1, triangleY1, triangleX2, triangleY2, triangleX3, triangleY3);
                    drawCount := drawCount + 1;
                    RefreshScreen();
                end;
    until WindowCloseRequested();;
```

If Statement

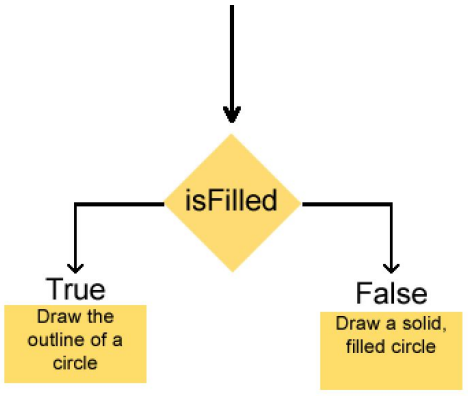
	Details / Answer
An if statement …	Is used to execute a section of code if a certain dependency has been met or not. True or False.
This chooses between …	Two possible branches that can be executed. One branch will be executed if the dependency is met, otherwise, the second branch will execute.

Example 1: To draw or fill a circle

This *if* statement is responsible for choosing between drawing a circle outline or drawing a solid circle in one of my programs. Here is what the statement looks like in code:

```
begin
    if isFilled = False then
        DrawCircle(clr, x, y, radius)
    else
        DrawCircle(clr, isFilled, x, y, radius);
end;
```

Flow Chart:

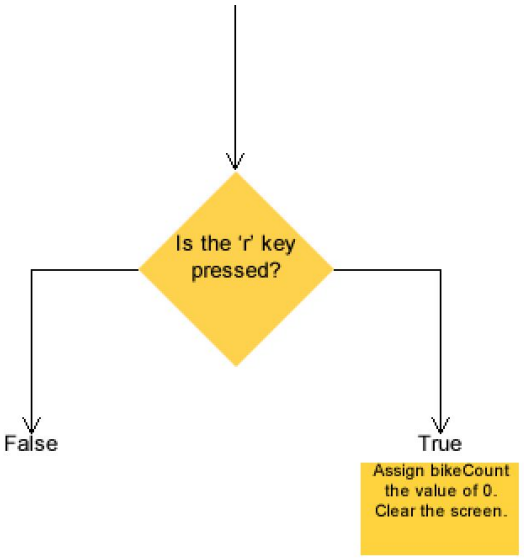


Example 2: Reset!

This *if* statement I've used is responsible for resetting a pre-test loop. It only has a branch that is executed if it is true. For the false value, nothing is executed.

```
if KeyDown(vk_r) then
    BikeCount := 0;
    ClearScreen(ColorWhite);
```

Flow Chart:



Case Statement

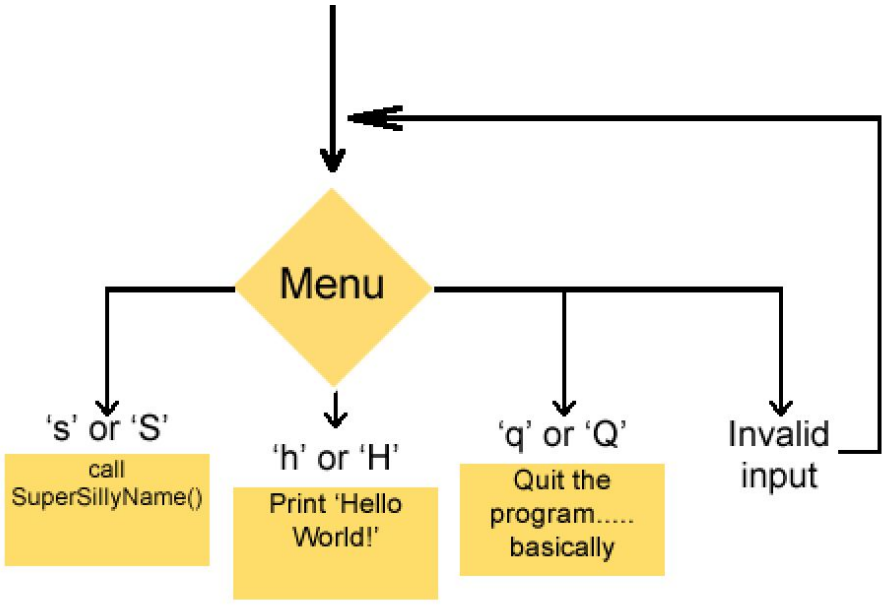
	Details / Answer
A case statement …	Unlike *if* statements where there are only two paths, case statements allow for a number of paths or branches to be chosen on depending on the condition.
This chooses between …	One of the defined branches depending on which condition is met.

Example 1: A small menu

This case statement is a small example of how a simple menu system can be implemented. Here is what the case statement looks like in code:

```
case option of
    's', 'S': SuperSillyName(name);
    'h', 'H': WriteLn('Hello World!');
    'q', 'Q': WriteLn('Peace...');
else
    WriteLn('Shiieeeet ',name, '. Please try again.');
```

Flow Chart:



Actions Performed

When a case statement is executed the computer performs the following steps:

1. The computer reads the possible paths associated with the case statement
2. The computer is instructed which path to follow based on user input
3. The instructions for that path are executed

While Loop

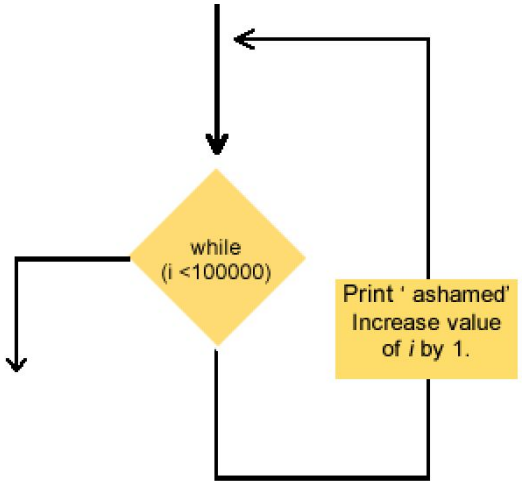
	Details / Answer
A while loop …	Is used to repeat a section of code while a dependency is being met.
This loop runs its body …	Well, it actually may not run its body at all if the pre-test condition is met, if it is not met, the body will repeat until it is.

Example 1: You should feel ashamed

This loop will print the word *ashamed* 100000 times! Refer to the code below for an understanding:

```
i := 0;
while (i < 100000) do
begin
    Write(' ashamed');
    i := i + 1;
end;

WriteLn(' of yourself.');
```

Flow Chart:



Actions Performed

When a while loop is executed the computer performs the following steps:

1. The computer reaches the *while* loop
2. The computer checks the condition of the loop
3. If the condition is met, the computer executes the loop until the condition no longer applies.
4. If the condition of the loop is not met, the computer does not run the loop at all.

Repeat Loop

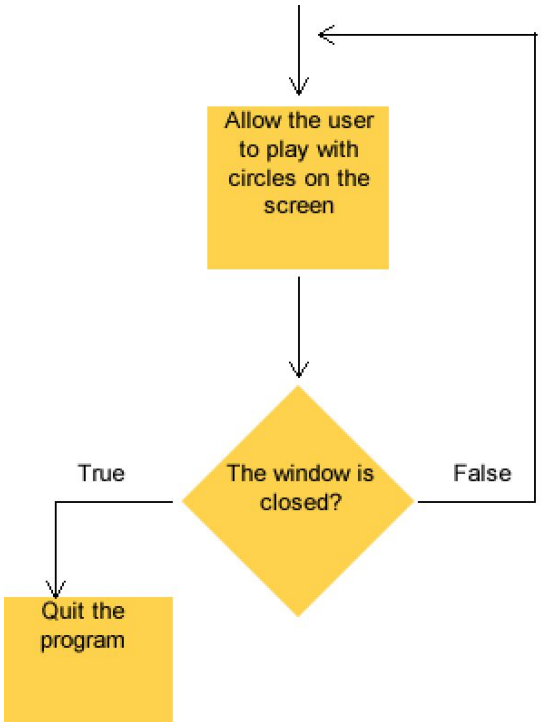
	Details / Answer
A repeat loop…	Is a section of code, which is executed at least once before any conditions are checked.
This loop runs its body…	At least once, unlike *while loops*, which check their condition before running, *repeat loops* always check their condition at the end.

Example 1: Play with some bikes

This is basically the beginning of a small game. This loop enables a user to move, enlarge and change the colour of a circle for as long as they want. The loop only finishes when the program window is closed. Here's the code below:

```
repeat
        ProcessEvents();
        ClearScreen(ColorWhite);
        if KeyTyped(vk_r) then
            circleX := Rnd(ScreenWidth());
        if KeyTyped(vk_r) then
            circleY := Rnd(ScreenHeight());
        if KeyTyped(vk_c) then
            clr := RandomColor;
        if MouseClicked(LeftButton) and PointInCircle(MouseX(), MouseY(), circleX, circleY, radius) then
            myCircleIsFilled := not myCircleIsFilled;
        if KeyDown(vk_Left) then
            circleX -= 3;
        if KeyDown(vk_Right) then
            circleX += 3;
        if KeyDown(vk_Up) then
            circleY -= 3;
        if KeyDown(vk_Down) then
            circleY += 3;
        if KeyDown(vk_a) then
            radius := radius + 3;
        if KeyDown(vk_z) then
            radius := radius - 3;
        if radius <= 10 then
            radius := 10;
        if radius >= 200 then
            radius := 200;
        if circleX < radius then
            circleX := radius;
        if circleX > ScreenWidth() - radius then
            circleX := ScreenWidth() - radius;
        if circleY < radius then
            circleY := radius;
        if circleY > ScreenHeight() - radius then
            circleY := ScreenHeight() - radius;
        DisplayCircle(clr, myCircleIsFilled, circleX, circleY, radius);
        RefreshScreen();
    until WindowCloseRequested();
```

Flow Chart:



Actions Performed

When a repeat loop is executed the computer performs the following steps:

1. Reaches the loop and executes all the instructions inside the loop
2. Once that loop has been run, the condition of the loop is checked
3. If the condition is met, the loop will finish
4. Otherwise, the loop will repeat until the condition has been met

Core Exercise 2 – Hand Execution:

```
function Func1(val1, val2: Integer) : Integer;
begin
    if val1 > val2 then result := val1 - val2
    else if val2 > val1 then result := val2 - val1
    else
        result := 0;
end.

Func1(5, 3)
Func1(1, 4)
Func1(5, 5)
```

```
procedure DisplayError(code: Integer) : Integer;
begin
    case code of
        400: WriteLn('Bad Request');
        401: WriteLn('Unauthorised');
        402: WriteLn('Payment Required');
        403: WriteLn('Forbidden');
        404: WriteLn('Not Found');
        else WriteLn('Unknown');
    end;
end.

DisplayError(402);
DisplayError(404);
DisplayError(500);
```

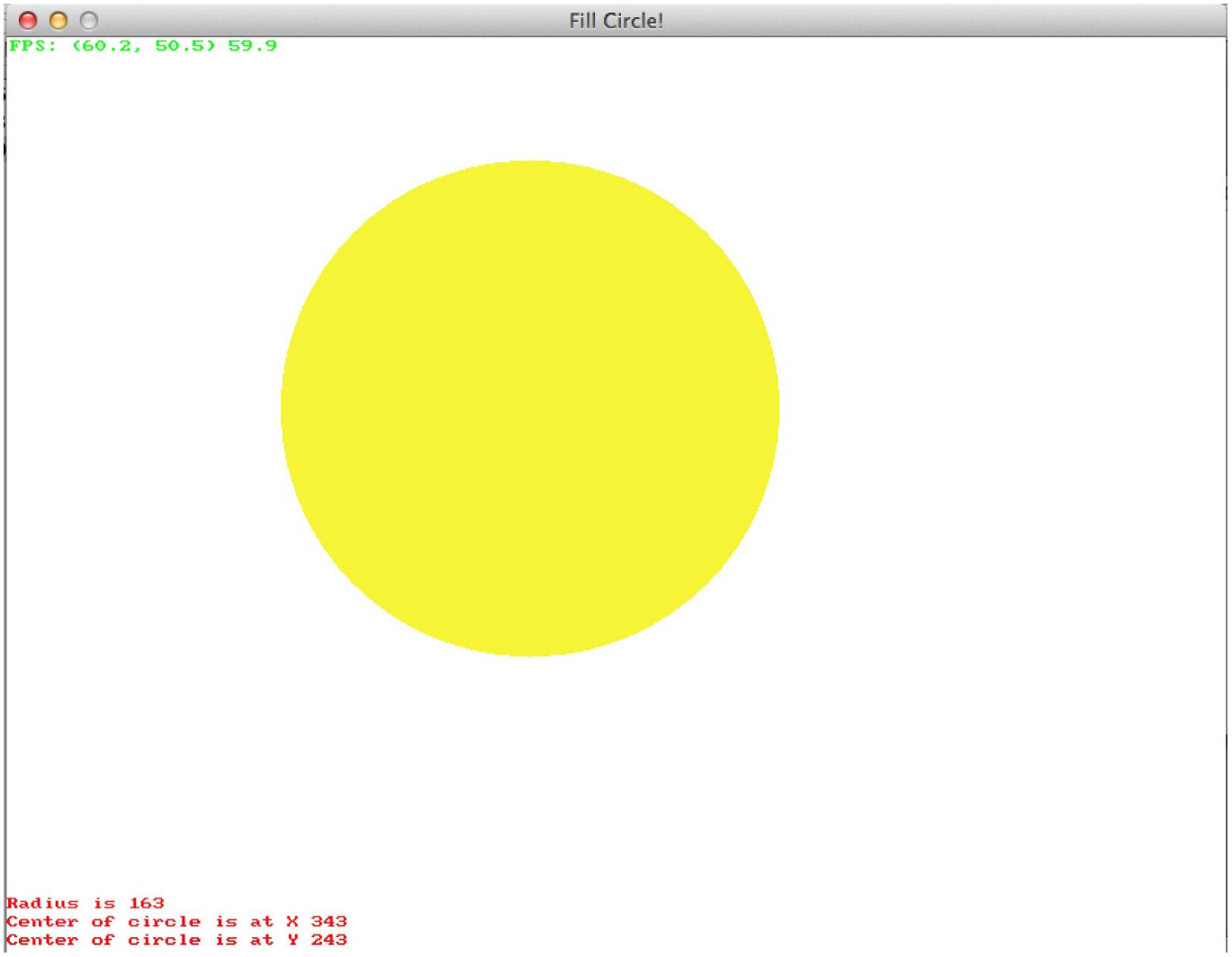
```
function Cost(age: Integer; weekend: Boolean): Integer;
begin
     if age <= 6 then result := 0
     else if (age <= 15) and weekend then result := 5
     else result := 10;
end;

Cost( 3, True )                Cost( 3, False )
Cost(14, True )                Cost(14, False )
Cost(20, True )                Cost(20, False )

function Func2(base, power: Integer) : Integer;
begin
    result := 1;
    while power > 0 do
    begin
         result := result * base;
         power := power - 1;
    end;
end.

Func2(2, 3)
Func2(3, 2)
Func2(15, 0)
Func2(12, 1)
```

```
function Func3(base, power: Integer) : Integer;
begin
    result := 1;
    repeat
         result := result * base;
         power := power - 1;
    until power <= 0;
end.

Func3(2, 3)
Func3(3, 2)
Func3(15, 0)
Func3(12, 1)
```

```
function Func4(val: Integer) : Integer;
var
    i: Integer;
begin
    result := 0;
    i := 1;
    while i <= val do
    begin
        if i mod 2 = 0 then // if it is divisible by 2
            result += 1;
        if i mod 5 = 0 then // if it is divisible by 5
            result -= 1;
        i := i + 1;
    end;
end.

Func4(2)
Func4(3)
Func4(15)
Func4(12)
```

Core Exercise 3 – Circle Drawing:

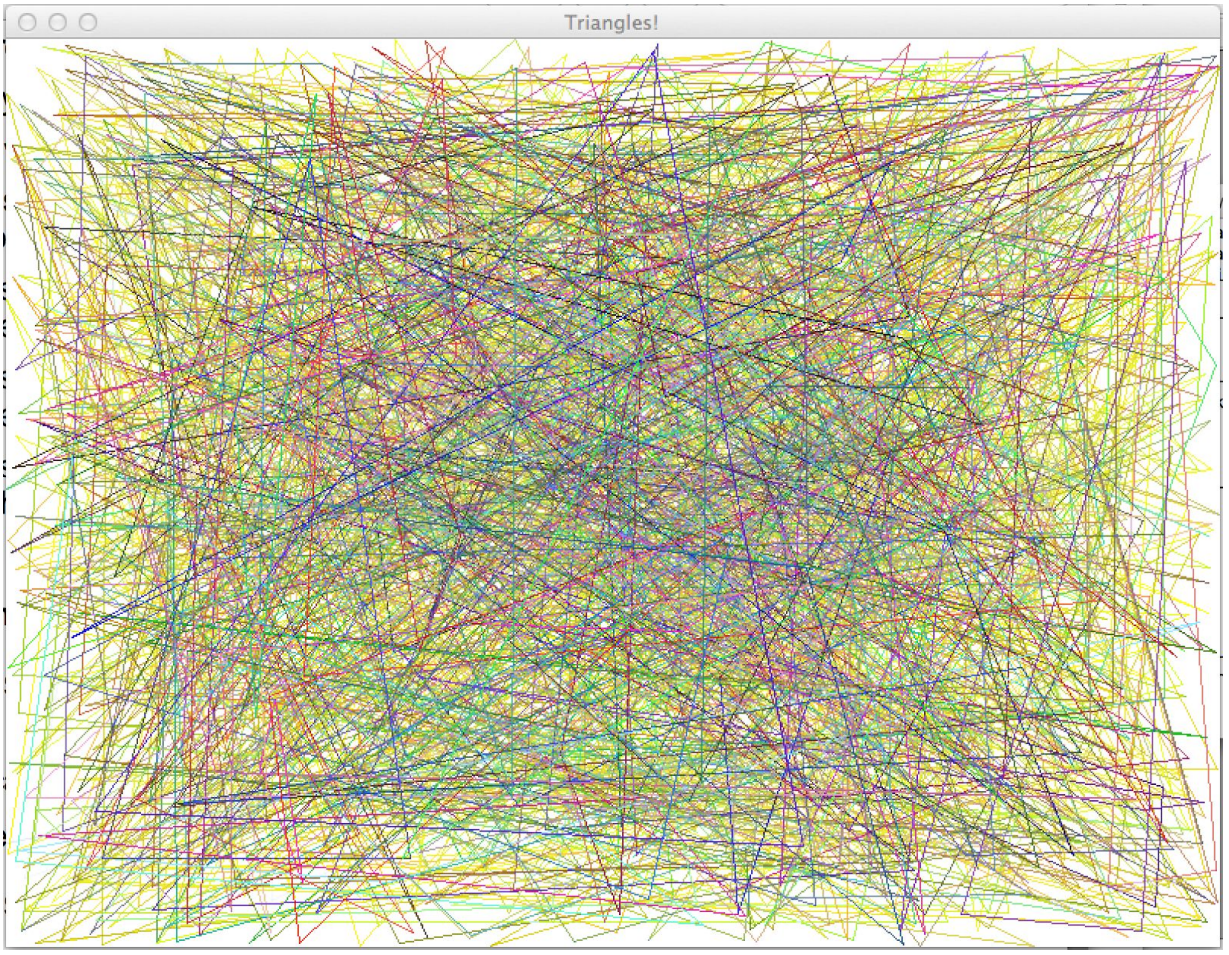
Here is a screenshot of my circle-drawing program!



Please refer to the code printout at the back for annotations.

Core Exercise 4 – Random Triangles:

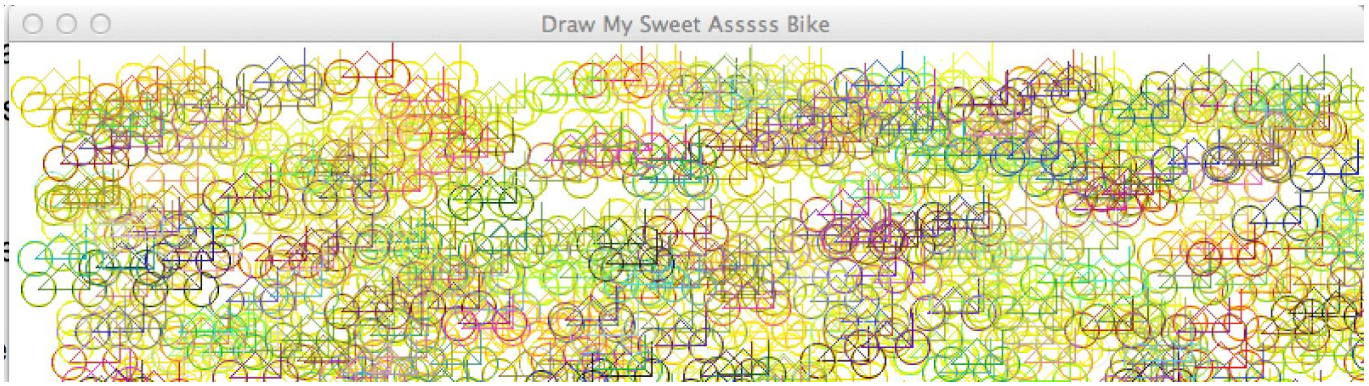
Here is a screenshot of my random triangles program!



You'll find the code for this application attached at the back!

Core Exercise 5 – Random Bikes:

Here is a screenshot of my random bikes program!



You'll find the code for this application attached at the back!

Core Exercise 6 – Boolean Expressions:

1. What are the values of the following boolean expressions?

	Expression	Given	Answer
a	`1 > 5`		
b	`1 < 5`		
c	`a > b`	`a := 1 and b := 2`	
d	`(a > b) or (b > a)`	`a := 1 and b := 2`	
e	`(a > b) or (b > a)`	`a := 2 and b := 2`	
f	`a or b or c`	`a := False, b := True, c := True`	
g	`(a or b) and (c or d)`	`a := False, b := True, c := False, d := False`	
h	`(a and b) or (c and d)`	`a := False, b := True, c := True, d := True`	
i	`a xor b`	`a := True, b := True`	
k	`not True`		
l	`(a or b) and (not (b and c))`	`a := True, b := False, c := True`	

2. Design an expression to check each of the following (invent variables as needed):

a	A person is older than 18
b	To ride a roller coaster you must be taller than 100cm
c	To ride the "adults only" roller coaster you must be older than 18 and taller than 100cm
d	A person is twice Fred's age, half his height, and has the same weight
e	Is the mouse within a rectangle? (*x, y, width, height*)
f	The user has clicked within rectangle