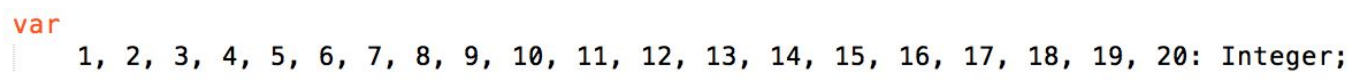
Introduction to Programming Glossary

By Reuben Wilson (9988289)

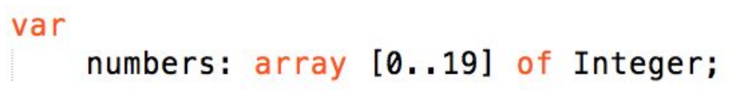
Programming Overview

Arrays are extremely effective in eliminating excessive variable creation and dealing with multiple values. Let's say I wanted to make 20 different variables, each one would store it's own unique number. In pascal, that would look like this:



```
var
    1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20: Integer;
```

It's long and messy and could even lead to difficulty in tracking variables and their values. By using an array, we can simplify the above stated variable creation so much that it looks like:



```
var
    numbers: array [0..19] of Integer;
```

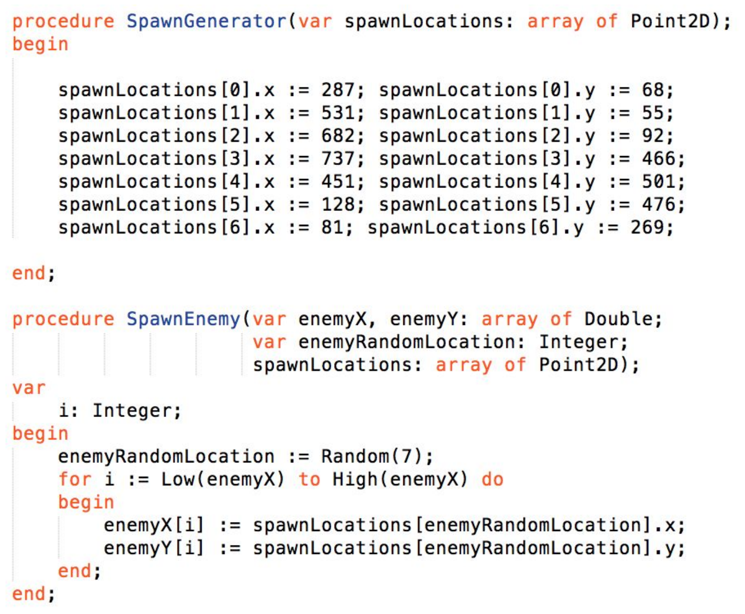
So, now we have an array called numbers, it can take 20 (0 to 19) unique values as integers, similarly to how the 20 declared variables previously could. It's neat and tidy and the values are easy to track!

So, I can relate and apply this understanding to the statistics program we made and appreciate that arrays made the overall program and it's functionality so much easier to implement because the program was dealing with multiple values, which would not have been efficiently dealt with if those values were implemented as their own, separate variables.

Iteration

For loops are used in conjunction with a loop counter, or a loop variable, which determines the amount of times the loop will be executed. For loops are often used in conjunction with arrays, where the length of the array is the value of the loop counter that the for loop uses. The best explanation I can offer is this snippet of code below with a picture diagram of how it works.

Pascal Code: This small example shows two procedures from my distinction project:



```
procedure SpawnGenerator(var spawnLocations: array of Point2D);
begin

    spawnLocations[0].x := 287; spawnLocations[0].y := 68;
    spawnLocations[1].x := 531; spawnLocations[1].y := 55;
    spawnLocations[2].x := 682; spawnLocations[2].y := 92;
    spawnLocations[3].x := 737; spawnLocations[3].y := 466;
    spawnLocations[4].x := 451; spawnLocations[4].y := 501;
    spawnLocations[5].x := 128; spawnLocations[5].y := 476;
    spawnLocations[6].x := 81; spawnLocations[6].y := 269;

end;

procedure SpawnEnemy(var enemyX, enemyY: array of Double;
                     var enemyRandomLocation: Integer;
                     spawnLocations: array of Point2D);
var
    i: Integer;
begin
    enemyRandomLocation := Random(7);
    for i := Low(enemyX) to High(enemyX) do
    begin
        enemyX[i] := spawnLocations[enemyRandomLocation].x;
        enemyY[i] := spawnLocations[enemyRandomLocation].y;
    end;
end;
```

How it works:

So, the for loop in the procedure SpawnEnemy is assigning each element in the arrays of enemyX and enemyY a random Point2D value based on the values provided by the array spawnLocations.

Array

	Details / Answer
An array is …	An array is a variable, which can store more than one value.
When I picture an array…	Like a normal variable, which has a container; but for arrays, there are multiple containers, or lots of containers in one large container.

Example 1: spawnLocations

This example array is responsible for storing the spawn locations for the enemies in my distinction project

Pascal Code:

```
spawnLocations: array of Point2D
```

Here are the values that that array has stored:

```
spawnLocations[0].x := 287; spawnLocations[0].y := 68;
spawnLocations[1].x := 531; spawnLocations[1].y := 55;
spawnLocations[2].x := 682; spawnLocations[2].y := 92;
spawnLocations[3].x := 737; spawnLocations[3].y := 466;
spawnLocations[4].x := 451; spawnLocations[4].y := 501;
spawnLocations[5].x := 128; spawnLocations[5].y := 476;
spawnLocations[6].x := 81; spawnLocations[6].y := 269;
```

Illustration:

Array Creation:
Let's say I create an array called names, which can take 7 values; the computer then allocates names as a variable, which can take 7 individual values. An analogy is that the computer places a container down called names and then puts 7 smaller containers inside the larger one.

For Loop

	Details / Answer
A for loop …	Allows code to be executed over and over again. They are used in conjunction with a loop counter, or in other words, a variable, which tracks the number of times the loop runs.
For loops are typically used with…	With arrays. I stated above that for loops use a kind of loop counter, think of the length of an array; this length value can be used as a counter value for a for loop.

Example 1: Print numbers
This example for loop prints every value in the array of numbers

Flow Chart:

Pascal Code:

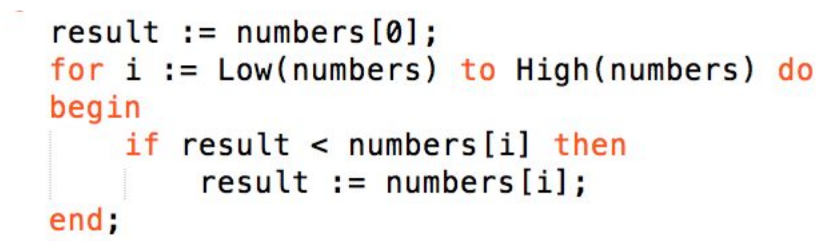
```
for i := Low(numbers) to High(numbers) do
begin
    WriteLn('You entered ', numbers[i]:4:2);
end;
```

Example 2: Assign result a value under circumstances

This for loop checks every value in the array numbers and assigns result the value of any value of number which is less than result.

Flow Chart:

Pascal Code:



```
result := numbers[0];
for i := Low(numbers) to High(numbers) do
begin
    if result < numbers[i] then
        result := numbers[i];
end;
```

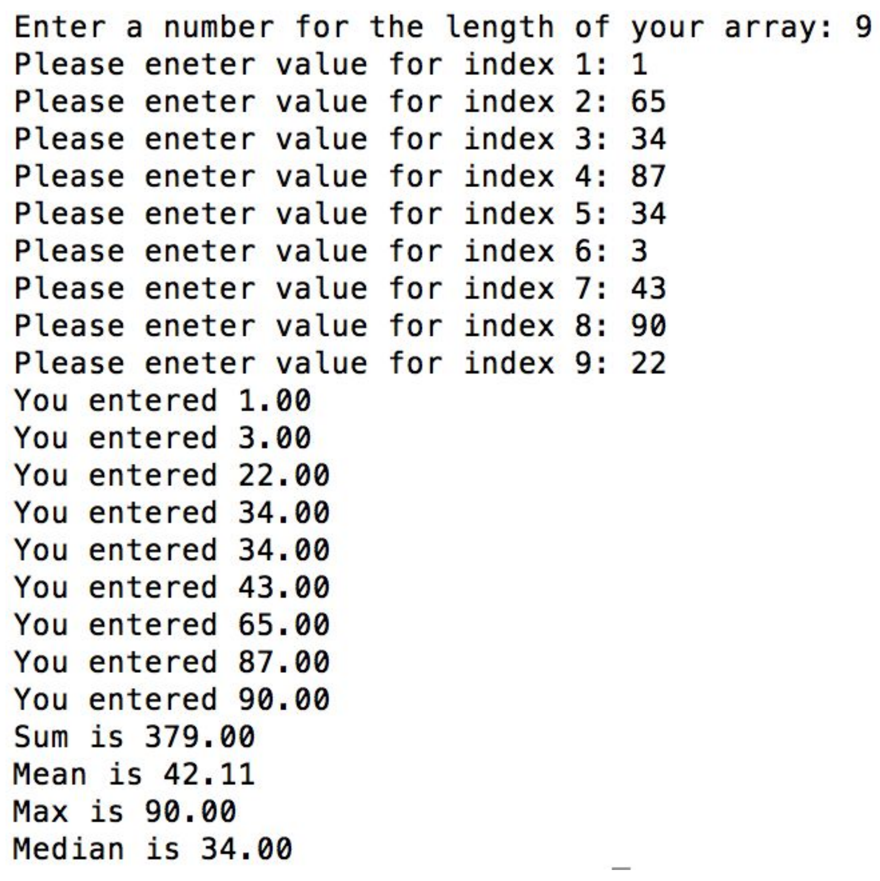
Actions Performed

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

1. The computer reaches the *for* loop
2. The computer then checks how many times the loop is to be iterated
3. The computer executes the loop for each iteration
4. For each execution, the computer utilizes the instructions within the loop
5. Once the iteration is finished, the loop is over

Core Exercise 2 & 3 – Complete Statistics Program:

Here is a screenshot of my statistics program running:

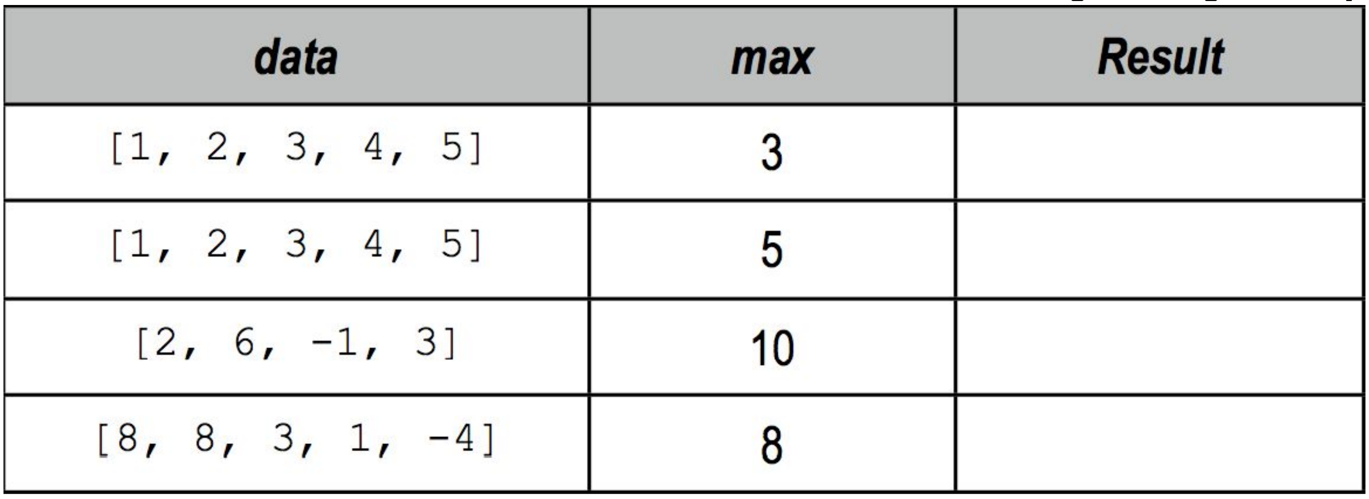


```
Enter a number for the length of your array: 9
Please eneter value for index 1: 1
Please eneter value for index 2: 65
Please eneter value for index 3: 34
Please eneter value for index 4: 87
Please eneter value for index 5: 34
Please eneter value for index 6: 3
Please eneter value for index 7: 43
Please eneter value for index 8: 90
Please eneter value for index 9: 22
You entered 1.00
You entered 3.00
You entered 22.00
You entered 34.00
You entered 34.00
You entered 43.00
You entered 65.00
You entered 87.00
You entered 90.00
Sum is 379.00
Mean is 42.11
Max is 90.00
Median is 34.00
```

Please find the code for this program attached below:

Core Exercise 4 – Hand Execution:

```
function ???(const data: array of Integer;
                        max: Integer): Boolean;
var
    i: Integer;
begin
    result := true;

    for i := Low(data) to High(data) do
    begin
        if data[i] > max then result := false;
    end;
end;
```



data	*max*	*Result*
`[1, 2, 3, 4, 5]`	3	
`[1, 2, 3, 4, 5]`	5	
`[2, 6, -1, 3]`	10	
`[8, 8, 3, 1, -4]`	8	

```
function ???(const data: array of Integer;
                    val: Integer): Boolean;
var
    i: Integer;
begin
    result := False;

    for i := Low(data) to High(data) do
    begin
        if data[i] = val then
        begin
            result := True;
            break;
        end;
    end;
end;
```

data	***val***	***Result***
`[1, 2, 3, 4, 5]`	3	
`[1, 2, 3, 4, 5]`	6	
`[2, 6, -1, 3]`	10	
`[8, 8, 3, 1, -4]`	8	

```
function Median (const data: array of Integer): Integer;
var
    startIdx, endIdx: Integer;
begin
    startIdx := Low(data);
    endIdx := High(data);

    while (startIdx <> endIdx) and (startIdx < endIdx) do
    begin
         startIdx := startIdx + 1;
         endIdx := endIdx - 1;
    end;

    if startIdx = endIdx then result := data[startIdx]
    else result := data[startIdx] + data[endIdx] / 2;
end;
```

data	*Expected Result*	*Actual Result*
`[1, 2, 3, 4, 5]`	3	
`[1, 2, 3, 4, 5, 6]`	3.5	
`[-2, -1, 0, 1, 2]`	0	
`[-2, -1, 1, 2]`	0	

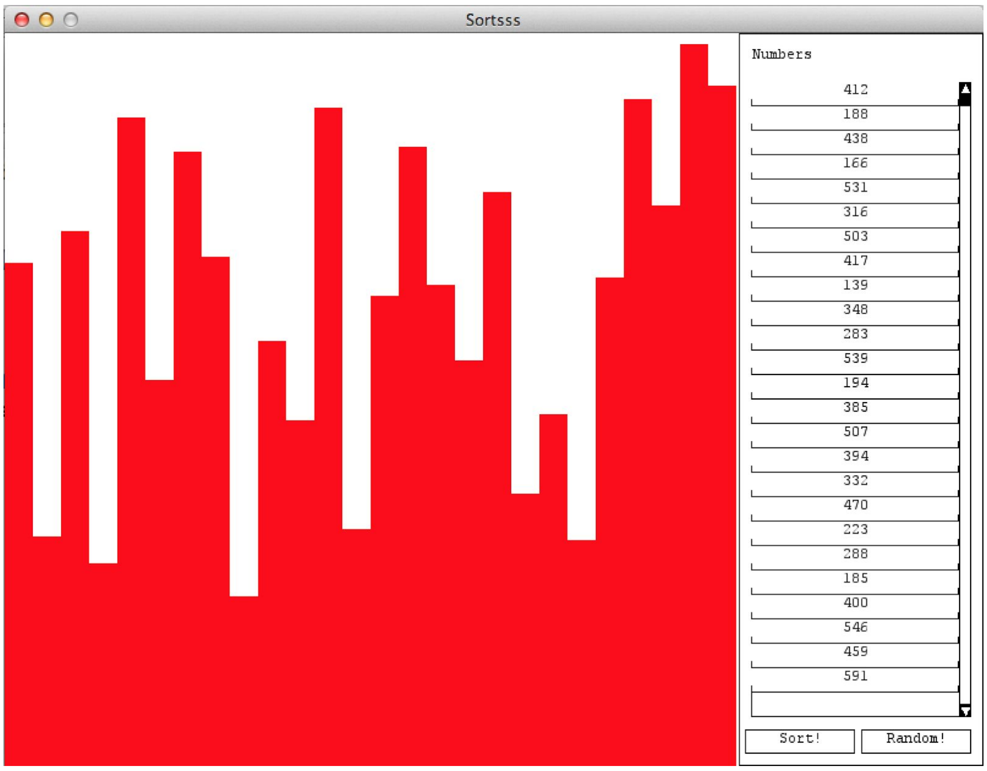
```
procedure ??? (var data: array of Integer;
          param2, param3: Integer);
var
    i: Integer;
begin
    for i := High(data) downto param2 + 1 do
    begin
         data[i] := data[i - 1];
    end;

    data[param2] := param3;
end;
```

initial data	*param2*	*param3*		*altered data*
`[1, 3, 3]`	1	2		
`[-1, 0, 1, 2, 3]`	2	6		
`[8, 4, 2]`	0	-1		
`[-2, -1, 1, 2]`	2	0		

Extension Exercise 1 – Sort Visualiser:

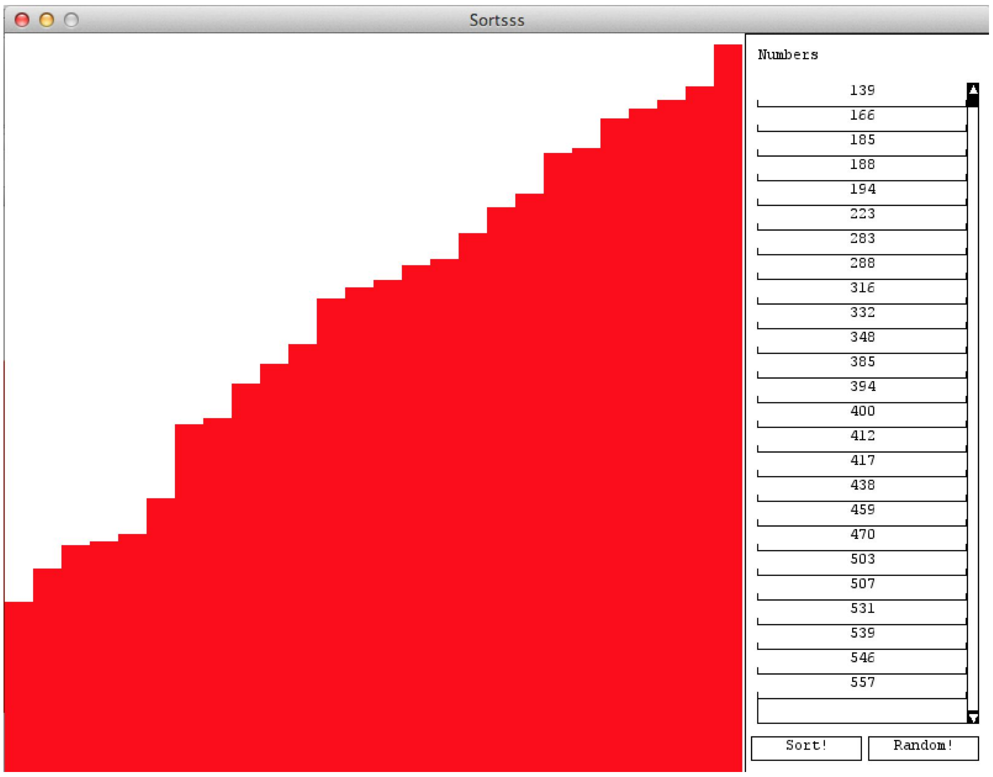
Here's some screenshots of my program running:

Before sort:



```
Numbers
412
188
438
166
531
316
503
417
139
348
283
539
194
385
507
394
332
470
223
288
185
400
546
459
591
```

After sort:

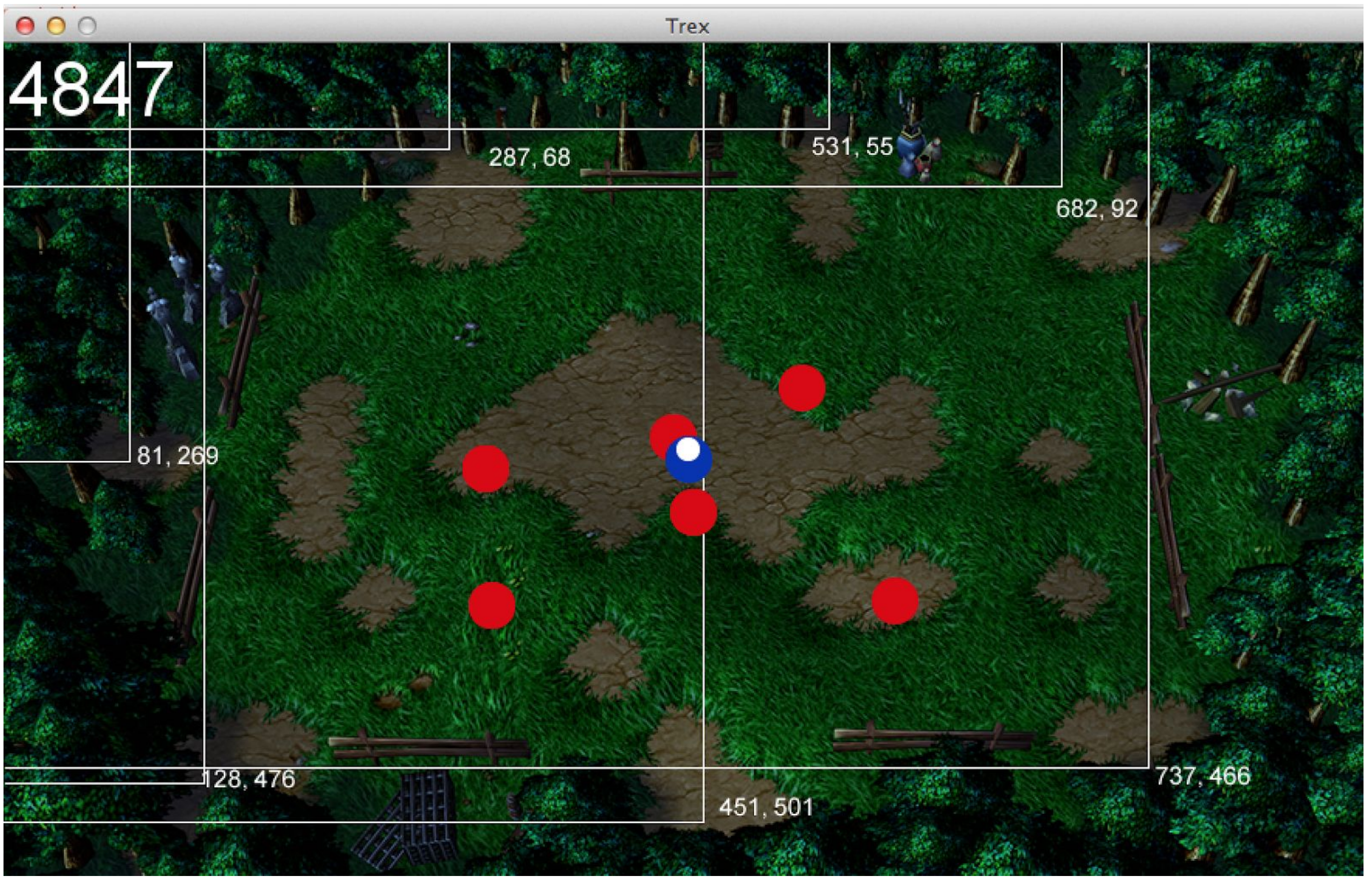


```
Numbers
139
166
185
188
194
223
283
288
316
332
348
385
394
400
412
417
438
459
470
503
507
531
539
546
557
```

You can click the random button to generate random numbers! Code below:

Extension Exercise 2 – Distinction Project Progress:

Here's a screen shot of my distinction project running, you can find all the code attached:


Trex
4847
287, 68
531, 55
682, 92
81, 269
128, 476
451, 501
737, 466