# ENGG1003 - Tuesday Week 2

## Calculating Pi
## C Arithmetic
## Datatypes

Brenton Schulz

University of Newcastle

February 21, 2019

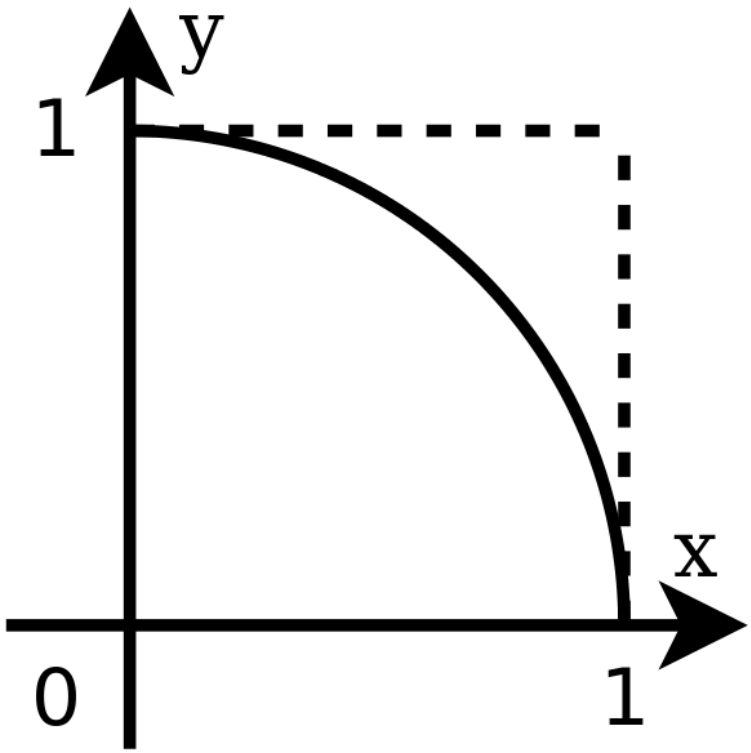
# Case Study: Calulating $\pi$

- Computers are *really* good at repetitive things
- Lets use this fact to calculate $\pi$ using a "monte-carlo" method
  - Informally, these are methods which solve problems by trying the same thing with different inputs until patterns emerge
  - It could repeat millions or billions of times
  - Name comes from the Monaco Principality's high concentration of casinos
- Algorithm pseudocode will be written before an implementation in C

# Case Study: Calulating $\pi$

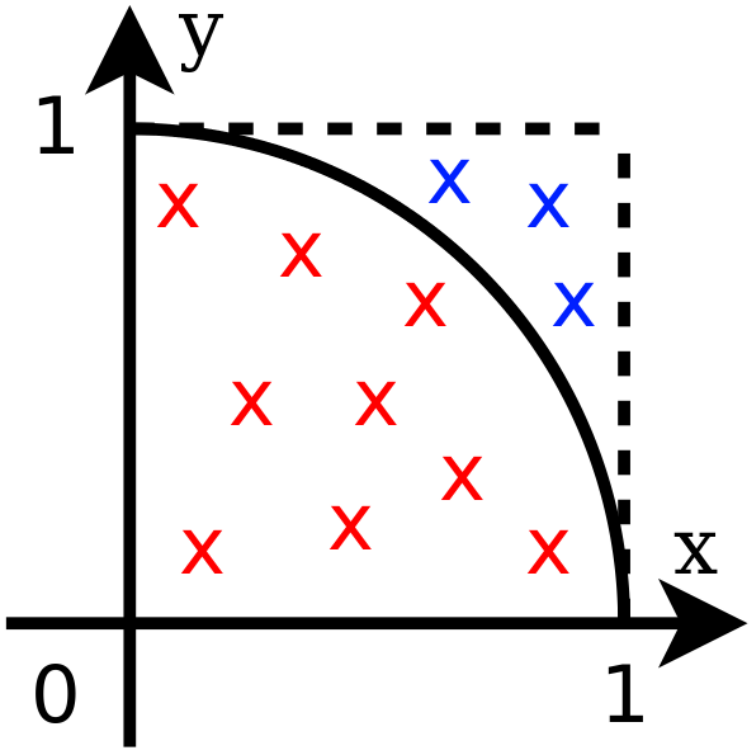
Consider a quadrant of a unit circle ($r = 1$) with a square around it:

- Area of the square $A_1 = 1$
- Area of the circle quadrant $A_2 = \frac{\pi r^2}{4} = \frac{\pi}{4}$
- Ratio of areas $\frac{A_2}{A_1} = \frac{\pi}{4}$
- Therefore $\pi = 4 \times \frac{A2}{A1}$



# Case Study: Calulating $\pi$

- We can't calculate the area ratio without knowing $\pi$
- Estimate it by:
  - Randomly picking many points inside the square
  - Test if the point is inside the circle with $x^2 + y^2 < 1$




- $\pi \approx 4 \times \frac{\text{Number of points which land inside circle}}{\text{Total number of points tested}} = 4 \times \frac{9}{12} = 3$

# Algorithm for Calculating $\pi$

- How can the above *mathematics* be turned into an *algorithm*?
- The algorithm needs to repeat the *same thing* multiple times
  - This implies use of a loop
- As the loop repeats, we need to keep track of the following *variables*:
  - The number of points tested
  - The number of points inside the circle
  - The $(x, y)$ coordinates of the point under test
- The incrementation of the "inside circle" count is *conditional* on the values of $x$ and $y$
  - This implies an IF...ENDIF flow control structure

# Algorithm for Calculating $\pi$

- The number of points tested will be an *integer*, we will call it $\mathrm{countTotal}$
- The number of points found to be inside the circle is also an integer, we will call it $\mathrm{countInside}$
- The point under test needs two "real" variables: $\mathrm{x}$ and $\mathrm{y}$
  - "Real" is the generic term for a number with integer and fractional components. Eg: 1.45
- The condition on incrementing $\mathrm{countInside}$ will be $x^2 + y^2 < 1$

# Algorithm for Calculating $\pi$

```
BEGIN
   integer countTotal = 0
   integer countInside = 0
   WHILE countTotal < A large number
      x = random number between 0 and 1
      y = random number between 0 and 1
      countTotal = countTotal + 1
      IF x^2 + y^2 < 1
         countInside = countInside + 1
      ENDIF
   ENDWHILE
   pi = 4*countInside/countTotal
   PRINT pi
END
```

# Missing Knowledge for C Implementation

- More information about arithmetic
  - Relational operators look useful
  - Is there a neat way to do $\mathrm{count}{=}\mathrm{count}{+}1$?
  - $\mathrm{countInside}$ and $\mathrm{countTotal}$ are both integers. What happens when we divide?
- Datatypes and how they are handled in arithmetic statements
- How do we generate random numbers?
- Syntax for WHILE loops and IF statements

# C Arithmetic

- Basic arithmetic was seen in the lab
  - You all did the lab, right?

| Operation | C Symbol |
|---|---|
| Addition | + |
| Subtraction | - |
| Multiplication | * |
| Division | / |

Table: Basic arithmetic operators in C

- Complex expressions can be built from these operators and parentheses

# C Arithmetic

## Examples:

$$z = x^2 + 5(y + b) \qquad \mathrm{z = x*x + 5*(y + b);}$$

$$u = \frac{x+1}{x-1} \qquad \mathrm{u = (x + 1)/(x - 1);}$$

$$v = z^3 + \frac{5(y+b)}{2} \qquad \mathrm{v = z*z*z+(5*(y + b))/2;}$$

- Multiplication is not assumed. If you write $5(\mathrm{y+b})$ the compiler will generate a syntax error.
- To be valid C expressions the semicolon is required.

# C Arithmetic

- C supports two time-saving *unary* operators:
  - Very useful in loops.

| Operation | C Syntax | Replaces |
|---|---|---|
| Increment | x++; or ++x; | x = x + 1; |
| Decrement | x–; or –x; | x = x - 1; |

- It also supports the following shorthand syntax:

| | |
|---|---|
| x = x + y; | x += y; |
| x = x - y; | x -= y; |
| x = x * y; | x *= y; |
| x = x / y; | x /= y; |

# C Arithmetic

What's the difference between `x++` and `++x`?

- `x++` is a *post-increment*
- `++x` is a *pre-increment*
- If they appear in an arithmetic expression, pre-increment is processed *before* the variable is used and post-increment is processed *after* it is used.
- In isolation there is no difference.

# Increment Example

```c
#include <stdio.h>
int main() {
   int x = 0;
   int y = 0;
   int z = 0;
   y = ++x + 10;
   printf("Pre-increment: %d\n", y);
   y = z++ + 10;
   printf("Post-increment: %d\n", y);
   return 0;
}
```

Pre/post-inc/decrements will be used when working with *arrays*, more details in coming weeks.

# Modulus

- Computers frequently only deal with integers
- Integer division in C ignores (truncates) any fractional component
- The *modulus* operator provides the remainder after division
  - Implemented with the $\%$ character
  - $\mathrm{a} \ \% \ \mathrm{b} =$ remainder of $\mathrm{a} \ / \ \mathrm{b}$
  - Very useful for tasks performed every *nth* loop
- Example:
  - $10 \ / \ 3 = 3$
  - $10 \ \% \ 3 = 1$

# Relational Operators

- C supports six *relational* operators:

| Operation | C Symbol |
|---|---|
| Less than | < |
| Less than or equal to | <= |
| Greater than | > |
| Greater than or equal to | >= |
| Equal to | == |
| Not equal to | != |

Table: Relational operators in C

# Relational Operators

- The result of a relational operation is $0$ or $1$
  - C treats $0$ as Boolean FALSE and non-zero as TRUE
- They are typically used as flow control conditions
  - $\text{if(condition) } \{\text{statements}\}$
  - $\text{while(condition) } \{\text{statements}\}$
- While we're here: the above is the correct syntax for IF and WHILE flow control in C

# Modulus Example 1 - Printing Every *nth* Loop

```c
#include <stdio.h>
int main() {
   int x = 0;
   while(x < 1000)
   {
      // Presumably something useful is done with x
      // inside this loop
      if(x%100 == 0)
         printf("%d\n", x);
   }
   return 0;
}
```

# Modulus Example 2 - Finding Factors

```c
#include <stdio.h>
int main() {
  int input;
  int x;
  printf("Enter an integer to factorise: ");
  scanf("%d", &input);
  x = input;
  while(x > 0) {
    if(input % x == 0) // ie: if the remainder is zero
      printf("%d is a factor of %d\n", x, input);
    x--;
  }
  return 0;
}
```

Observe that the while() loop loops over every value of x from input to 1. We will discuss a flow control method designed for this (the for loop) later.

# C Arithmetic Operator Precedence

- C has an "order of operations"
- eg: $1+5*2$ evaluates to $11$
- Multiplication and division first
- Addition and subtraction second
- Relational operators somewhere below that
- If in doubt: force order with parentheses
  - This makes the code more readable
  - It doesn't cost you anything
  - C compilers understand algebra and will optimise inefficient expressions automatically

# Data types

- You have hopefully noticed that all variables are *declared* before use
- Declaration specifies the variable's:
  - Datatype
  - Name
  - An *initialisation value* (optional)
    - Always assume uninitialised variables have random values! Behaviour varies between compilers and target platforms.
- C is a "strongly-typed" language
  - Every variable has a fixed type

# Integer Data types

- Integer data types vary by their:
  - Size
  - Support for negative numbers
- C integer types can be 1, 2, 4, or 8 bytes long
- Each type can be *signed* or *unsigned*
  - Unsigned number are never negative but you get double the value range

TODO: Keep hacking until we've implemented $\pi$.