# ENGG1003 - Tuesday Week 2

## Examples

Brenton Schulz

University of Newcastle

February 27, 2020

# Example: Testing Factors

Task: Write a C program which reads two integers from the user and tests if the second is a factor of the first, printing the result.

Eg: If the user enters 12 and 3 the program would print something like:

```
3 is a factor of 12
```

# Example: Testing Factors

Lets convert the problem down into “high level” pseudocode:

```
BEGIN
  integer x
  integer y
  READ x from the user
  READ y from the user
  Test if y is a factor of x
  Tell the user the result
END
```

Can every line of pseudocode be turned into C? How are we going to test if one number is a factor of another?

# Example: Testing Factors

Lets make an attempt at factor testing, perhaps start with:

Definition: *Factors* are numbers we can multiply together to make another number. Eg: 2 and 3 are factors of 6 as $2 \times 3 = 6$.

Is this definition useful for this problem? Is it easy to turn this definition into C code?

# Example: Testing Factors

No, not really. C can't easily do "can I find another integer that multiplies with $y$ to make $x$". That instruction is not *executable*.

Lets try again:

An integer, $y$, is a factor of a number, $x$, if the integer evaluation of $x \div y$ has no remainder.

Can *this* become C code?

# Example: Testing Factors

YES! We can use the modulus operator, %, to test if a division has no remainder with the code:

```
if( (x % y) == 0) {
  // y is a factor of x
}
```

# Modulus Example - Factor Testing

With this fact, lets tweak the pseudocode:

```
BEGIN
  integer x
  integer y
  READ x from the user
  READ y from the user
  IF (x % y) == 0
    PRINT y is a factor of x
  ELSE
    PRINT y is NOT a factor of x
  ENDIF
END
```

# Modulus Example - Factor Testing

...and convert each line to C:

(`printf();` output changed to fit on slide)
(C Code is almost missing `#include <stdio.h>`)

| Pseudocode | C Code |
|---|---|
| `BEGIN` | `int main() {` |
| `  integer x` | `  int x;` |
| `  integer y` | `  int y;` |
| `  READ x from the user` | `  scanf("%d", &x);` |
| `  READ y from the user` | `  scanf("%d", &y);` |
| `  IF (x % y) == 0` | `  if( (x % y) == 0) {` |
| `    PRINT y is a factor` | `    printf("%d is a factor\n", y);` |
| `  ELSE` | `  } else {` |
| `    PRINT y isn't a factor` | `    printf("%d isn't factor\n", y);` |
| `  ENDIF` | `  }` |
| `END` | `}` |

# Modulus Example - Code with Prompt

```c
#include <stdio.h>
int main() {
  int x, y;
  printf("Enter an integer: ");
  scanf("%d", &x);
  printf("Enter another integer: ");
  scanf("%d", &y);
  if(x % y == 0) { // ie: if the remainder is zero
    printf("%d is a factor of %d\n", y, x);
  } else {
    printf("%d is NOT a factor of %d\n", y, x);
  }
  return 0;
}
```

Listing 1: `factorTest.c`

# Factor Testing Discussion

- Is this code *robust*?
- Can the user enter numbers which make the code produce the wrong number?
- What happens if $y > x$?
  - It might be fine, it might not. Have a think about it and do some testing in the lab
- Get in the habit of testing code, both with "expected" input and "weird" input
- What happens if you enter letters instead of numbers? Or negatives? Or ask where the bathroom is?

# Confused so Far?

# PASS would be a good thing to attend

| | | |
|---|---|---|
| Monday | 10-11pm | HE28 |
| Wednesday | 11-12pm | HC02 |
| Thursday | 12-1pm | RW219 |

# Modulus Example 2 - Factorisation

- We now know how to test if a number is a factor of another
- What about a full factorisation?

**Task:** Write a C program which outputs all of the factors of a given integer.

# Modulus Example 2 - Factorisation

- How does factorisation happen, anyway?
  - Normally? In your head. “Dream up” the answer.
  - On a computer? We can *brute force* it.
    - ie: Simply test every integer which might work
  - All factors of a number, $k$, are in the range [1,k]
    - 1 and k are always factors, so explicitly testing them is optional
  - They are also all integers
  - Thankfully, this is a finite number of tests
- Faster algorithms probably exist
  - You would need to consult number theory literature
  - This is beyond my knowledge

# Modulus Example 2 - Factorisation

- How can we test lots of numbers?
  - The program needs to count
  - The input is unknown, so we need a loop
  - We can't "hard code" counting when we don't know when counting needs start and/or stop!

# Modulus Example - Factorisation

Lets write some pseudocode for the factorisation problem. We start with something really “high level”:

```
BEGIN
  Integer x
  READ a value for x from the user
  Calculate x's factors
  PRINT x's factors
END
```

# Modulus Example - Factorisation

Now lets *imply* a loop, but not explicitly write it:

```
BEGIN
 Integer x
 READ a value for x from the user
 Test if every integer from 1 to x is a factor of x
 PRINT x's factors
END
```

# Modulus Example - Factorisation

- How do we code "Test if every integer from 1 to x is a factor of x"?
- Well, we know how to test one integer, `n`:
  `if( x % n == 0)`
- How do we *count?*
- With what we know so far? A `while()` loop!

```
integer count = 1
WHILE count <= n
  count = count + 1 // Counts from 1 to n
ENDWHILE
```

- A "better" method will be shown later with `for(;;)` loops

# Modulus Example - Factorisation

```
BEGIN
  integer x
  integer count = 1
  READ a value for x from the user
  WHILE (count <= x)
    IF (x % count) == 0
      PRINT <count> is a factor of <x>
    ENDIF
    count = count + 1
  ENDWHILE
END
```

# Modulus Example - Factorisation

- Notice the `PRINT` statement *inside* the loop
  - Previous pseudocode has factorisation and printing as different steps
- This means we don't need to remember a list of factors as we go
- We will learn how to work with lists later
  - C calls a list of variables an *array*
- Lets read and run the final program...

# Modulus Example - Factorisation

```c
#include <stdio.h>
int main() {
  int count = 1, x;
  printf("Enter an integer to factorise: ");
  scanf("%d", &x);
  while(count <= x) {
    if(x % count == 0) // If the remainder is zero
      printf("%d is a factor of %d\n", count, x);
    count++;
  }
  return 0;
}
```

Listing 2: `factors.c`

# Modulus Example - Factorisation

Example output:

```
Enter an integer to factorise: 76545478
1 is a factor of 76545478
2 is a factor of 76545478
38272739 is a factor of 76545478
76545478 is a factor of 76545478
```

Is it correct? Check output with Wolfram Alpha

Observation: A modified version of this code (with `unsigned int`) takes around 15 seconds to factorise 4294967294

# Discussion

- Pay close attention to the value of `count`
- It is initialised to 1
- It is used *before* incrementing it
- Incrementing is the last thing in the loop
- The loop *condition* is “less than *or equal to*” so that `x` itself is explicitly tested as a factor
  - Remember that 1 and x are always factors of x?
  - This method is not optimised - speed improvements exist

# Square Root Algorithm

- The square root lab task includes the following formula:
$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{n}{x_k}\right) \tag{1}$$
- Last year this expression confused many

# Square Root Algorithm

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{n}{x_k}\right) \tag{2}$$

- It describes how $x$ changes with "time"
- $x$ "now" is $x_n$
- $x$ in the future is $x_{n+1}$
- The equation describes how the $=$ symbol behaves on a computer

# Another Simple Example

$$x_{k+1} = x_k + 1 \tag{3}$$

- In C: `x = x + 1`
- Before the line is executed `x` has a value
- The calculation of `x + 1` is done
- The result then becomes the *new* value of `x`

# Square Root Algorithm

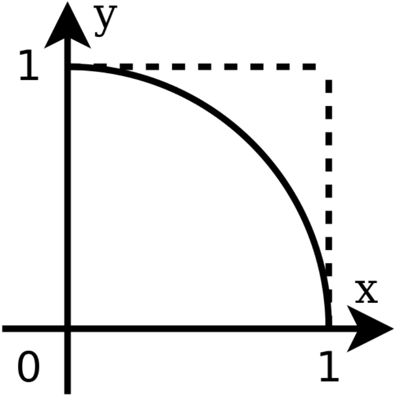
$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{n}{x_k}\right) \tag{4}$$

- In C: `x = 0.5*(x + n/x)`
- Before the line is executed `x` has a value
- The calculation of `0.5*(x+n/x)` is done
- The result then becomes the *new* value of `x`
- Write this code in the lab

# Case Study: Calulating $\pi$

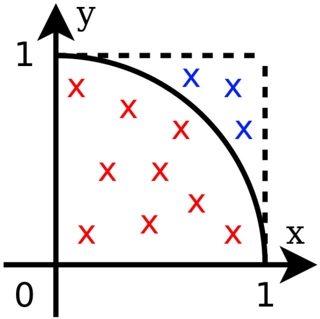
Consider a quadrant of a unit circle $(r = 1)$ with a square around it:

- 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 $\pi$

```
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*x + y*y < 1
      countInside = countInside + 1
    ENDIF
  ENDWHILE
  pi = 4*countInside/countTotal
  PRINT pi
END
```