ENGG1003 - Friday Week 1

Algorithms and Pseudocode

Brenton Schulz

University of Newcastle

March 1, 2019

Lab feedback? Yell it out to me.

Algorithms

- Informally, an *algorithm* is a series of steps which accomplishes a task
- More accurately, the steps (instructions) must:
  - Have a strict order
  - Be unambiguous
  - Be executable
- "Executable" means that the *target platform* is capable of performing that task.
  - eg: An industrial welding robot can execute "move welding tip 1 cm left". A mobile phone can't.

Algorithms

- An algorithm exists purely as an abstract concept until it is communicated
- We will use:
  - *Pseudocode* to communicate algorithms to ourselves and other people
  - The languages C and MATLAB to communicate algorithms to computers
- Pseudocode *can* be very formal, but as engineers we will only use formal rules if required
  - eg: When documenting algorithms for other people
  - Your own "working out" can be anything that helps *you*

Algorithm Example 1

Name: Algorithm given to mum to start my car (2015 Tarago)
Result: The vehicle's engine is idling
Initialisation: stand next to the vehicle, key fob in hand

1. Depress the unlock button on the key fob, car will beep twice
2. Place key fob in your pocket
3. Enter the vehicle, sit in the driver's seat
4. Ensure that the gear selector has P engaged
5. Depress the brake pedal
6. Press the engine start button
7. Wait 5 seconds
8. If engine is not idling
    - Call me

Example Discussion

- Algorithms typically need to feel over-explained
  - Computers are *really stupid*; get in the habit of over-thinking everything
- The algorithm contained *flow control* in the form of an "if" statement
  - The final step ("call me") was *conditional* on the car not starting
- We will discuss conditional logical statements later, but first...

Algorithm Example 2

A wife asks her husband, a programmer, “Could you please go shopping for me and buy one carton of milk, and if they have eggs, get 6?

A short time later the husband comes back with 6 cartons of milk and his wife asks, “Why did you buy 6 cartons of milk?

He replies, They had eggs.

Algorithm Example 2a

Lets make this more realistic.

A wife asks her robot helper, “Could you please go shopping for me and buy one carton of milk, and if they have eggs, get 6?

The robot replies: “Unknown instruction: ‘get 6’. ”

Flow Control

- Instructions in an algorithm execute in an ordered list
    - ie: top to bottom
- Flow Control is any algorithmic mechanism which changes the default “top to bottom” execution behaviour
- We will discuss IF statements and *loops*
- Flow control (almost) always requires a *condition*

Conditions

- Computers don't understand "maybe"
- A *condition* must be absolutely **true** or **false**
- Human examples:
  - I am within the boundary of the Callaghan campus
  - I am alive
  - My net worth is below AU$100M
- Computer examples:
  - `i` is less than 184
  - `x` plus `y` is not equal to zero
  - Input data has been given to the program
  - A division by zero has occurred

Code Blocks

- A *block* is a set of instructions which are, for some reason, grouped together
- If a single condition controls multiple instructions they can go together in a block
- A block is typically indicated via indentation
- Eg:

```
IF it is raining
  Pack an umbrella
  Drive to campus instead of walking
  Leave home 40mins early to find parking
ENDIF
```

IF Variants

- There are several versions of IF flow control:
  - IF ... ENDIF
  - IF ... ELSE ... ENDIF
  - IF ... ELSEIF ... ENDIF
- The IF and ELSEIF keywords indicate conditions
- The ELSE keyword is *unconditional*
- Which one you choose depends on need
  - Is there one thing which is conditional?
  - Do I need to make a choice between two or more options?
  - Could nothing be executed?

IF Statement Syntax

- The IF … ENDIF syntax is:

```
IF condition
  do some things
ENDIF
```

- Likewise: IF … ELSEIF … ENDIF syntax is:

```
IF condition1
  do some things
ELSEIF condition2
  do other things
ENDIF
```

- And finally:

```
IF condition
  do some things
ELSE
  do some things
ENDIF
```

IF ... ELSEIF

- The IF ... ELSEIF construct can have multiple ELSEIF sections
- A *crucial* point:
    - Conditions are only tested *if the previous ones fail*
    - Once a condition is TRUE the others are ignored
    - ie: IF - ELSE implements a choice priority

Algorithm Example 3 - Quadratic Root Finding

From high school you should know that the equation

$$ax^2 + bx + c = 0 \tag{1}$$

has solutions given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

lets write an algorithm which provides real valued solutions to a quadratic equation.

Algorithm Example 3 - Quadratic Root Finding

Input: Real numbers a, b, and c
Output: Three numbers:

1. The number of solutions, N
2. One of the roots, x_1
3. The other root, x_2

Behaviour:

- If N is 2 then x_1 and x_2 are different real numbers
- If N is 1 then x_1 is the unique solution and x_2 is undefined
- If N is 0 then x_1 and x_2 are undefined

Algorithm Example 3 - Quadratic Root Finding

```
BEGIN
  INPUT: a, b, c
  D = b^2 - 4ac
  IF D < 0
    N = 0
  ELSEIF D = 0
    N = 1
    x1 = -b/(2a)
  ELSEIF D > 0
    N = 2
    x1 = (-b + sqrt(D))/(2a)
    x2 = (-b - sqrt(D))/(2a)
  ENDIF
END
```

- Reasonably formal pseudocode
- The IF ... ELSE IF flow control construct forces exclusive execution of only *one* block
- The first condition that is true causes execution of that block
- Subsequent blocks ignored
- Contains 3 *conditions*

Boolean Algebra Basics

- What if we want more complicated conditions? Boolean algebra is needed!
- Boolean algebra (or Boolean logic) is a field of mathematics which evaluates combinations of *logical variables* as either true or false
- Boolean *variables* can only take the values **true** (or 1) or **false** (or 0)
- Boolean algebra defines three *operators*:
    - OR
    - AND
    - NOT

Boolean Algebra Basics

- Boolean variables can be allocated any symbols (just like in "normal" algebra)
  - Typically get upper-case letters
  - eg: X = A OR B
- Various symbols can be used for OR/AND/NOT, we will only use the words here
  - Write them in capitals to remove ambiguity
  - C and MATLAB have their own symbols for Boolean algebra
  - Other courses (eg: ELE17100) will use different symbols again

Boolean Operators

- An *operand* is a value on which a mathematical operation takes place
  - eg: In "$1 + 2$" the 1 and 2 are operands and $+$ is the operator
- OR - Evaluates true if either operand is true
  - $X = A$ OR B
  - X is true if A or B is true
- AND- Evaluates true only when *both* operands are true
  - $X = A$ AND B
  - X is true only if both A and B is true

Boolean Operators

- OR and AND are *binary* operators
  - They operate on two operands
  - From Latin "bini" meaning "two together"
- The NOT operator is *unary*
  - It only operates on *one* operand
  - NB: The operand could be a single variable or complex expression
- NOT performs a logical inversion
  - NOT true = false
  - NOT false = true

Boolean Condition Examples

- My car needs a service if, since the last service, (more than 6 months has past) OR (more than 15000km have been travelled)
- You will pass this course if (you score 40% or more in the final exam) AND (the weighted sum of all assessments is more than 50%)
- A computer program repeats an algorithm if (there is still data to process) AND (errors have not occurred) AND (NOT (the user has terminated the program))

Algorithm Example 4 - Boolean Conditions

Problem: How can trigonometric functions be calculated by a computer?

One Solution: Series expansion! (Seen in MATH1120).

The function $\cos(x)$ can be evaluated with arithmetic as:

$$\cos(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} = 1 + \frac{-x^2}{2!} + \frac{x^4}{4!} + \frac{-x^6}{6!} + \frac{x^8}{8!} \ldots \quad (3)$$

Evaluation of this series needs two things:

1. The *loop* flow control concept
2. Some kind of *stop condition*

Factorial

- Do we know what factorials are?
- The product of an integer and all the integers between it and 1
- Notation in *mathematics* is the ! symbol
  - eg: $4! = 4 \times 3 \times 2 \times 1 = 24$
- **NB:** C uses ! for Boolean NOT
  - C does not have a factorial operator

Algorithm Example 4 - Boolean Conditions

- Computers can't count to infinity, we need to know when to stop
- Computers have limited precision, 16 significant figures is typical
- Observe that as k increases in Equation 4 the denominator increases *really fast* (4!=24, 10!=3628800)
- This implies that the value of $\frac{(-1)^k x^{2k}}{(2k)!}$ tends to drop as k increases
- Therefore, we can add terms until they are "too small"
- A maximum value of k can also be specified for safety

$$\cos(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} \tag{4}$$

Loops

- A *loop* causes an algorithm to execute a given block of instruction multiple times
- Loops typically require an *exit condition*
    - Without an exit condition they are called *infinite loops*
        - Yes, these have a purpose
- Multiple types of loops
    - WHILE *condition*...ENDWHILE
    - DO...WHILE *condition*
    - FOR *counter* FROM 1 TO *something*

Algorithm Example 4 - Boolean Conditions

- The *while* loop repeats a block of steps until the *condition* becomes false.

```
BEGIN
  INPUT x
  tmp = 1
  k = 0
  sum = 0
  WHILE (k<10) AND ((tmp>-1e-6) AND (tmp<1e-6))
    tmp = $\frac{(-1)^k x^{2k}}{(2k)!}$
    sum = sum + tmp
    k = k + 1
  ENDWHILE
END
```

- Here it loops until 10 *iterations* have occurred OR a precision limit is reached

This is where we left off...

Updates:

- Added “1+” to the series expansion
- Changed series expansion loop condition to be (absolute value(tmp) $<$ 1e-6)
  - Implemented with tmp greater than -1e-6 AND less than 1e-6 (ie: within the interval (-1e-6, 1e-6))

Loop Details

- WHILE conditions are tested before ''entering''
- The condition is tested before every repeat
- Variables in the condition should change inside the loop
    - Try to avoid infinite loops *unless you want one*
- What if we want to force the loop to execute *at least once*?

DO ... WHILE

- Same as WHILE except executes *at least once*
- The condition is tested at the end
- Loops repeats if condition is TRUE
- Syntax:

```
DO
  stuff
WHILE condition
```

FOR Loops

- A FOR loop executes a given number of times
- Used when the number of loop repeats is known *before* entering the loop
    - Repeat count could be “hard coded” as a number
    - Could also be a variable
- Can be easier to read than WHILE
- Example pseudocode syntax:

```
FOR x = 1 to 10
  Do something ten times
ENDFOR
```

- The *loop variable* is automatically incremented

BREAK Statements

- Sometimes you want to exit a loop *before* the condition is re-tested
- The flow-control mechanism for this is a BREAK statement
- If executed, the loop quits
- BREAKs typically go inside an IF to control their execution

FOR Example 1

- Two equivalent ways to implement the $\cos()$ series from before are:

```
BEGIN
  INPUT x
  sum = 0
  FOR k = 0 to 10
    tmp = $\frac{(-1)^k x^{2k}}{(2k)!}$
    sum = sum + tmp
    IF tmp < 1e-6
      BREAK
    ENDIF
  ENDWHILE
END
```

```
BEGIN
  INPUT x
  tmp = 1
  k = 0
  sum = 0
  WHILE (k<10) AND (tmp>1e-6)
    tmp = $\frac{(-1)^k x^{2k}}{(2k)!}$
    sum = sum + tmp
    k = k + 1
  ENDWHILE
END
```

GOTO

- There exists a GOTO flow control mechanism
  - Sometimes also called a *branch*
- It "jumps" from one line to a different line
- It exists for a purpose
- That purpose does not (typically) exist when writing C code
  - C *supports* a `goto` statement
  - It results in "spaghetti code" which is hard to read
  - Don't use it in ENGG1003
- You *can* use GOTO in ELEC1710

FOR Example 2 - Factorials

- Use FOR to count from 2 to our input number
- Keep a running product as we go

```
BEGIN
  INPUT x
  result = 1
  FOR k = 2 TO x
    result = result * k
  ENDFOR
END
```

- Is this algorithm robust? What happens if:
  - x = -1
  - x = 1
  - x = 0 (**NB:** 0! = 1 because *maths*)

Lets end with a joke