CLASS 7: INTRO TO ALGORITHMIC THINKING & FLOWCHARTS

ENGR 102 – Introduction to Engineering

Algorithmic Thinking

Algorithmic Thinking

- ***Algorithmic thinking***:
  - The ability to identify and analyze problems, and to develop and refine algorithms for the solution of those problems
- ***Algorithm***:
  - Detailed step-by-step procedure for the performance of a task
- Learning to program is about developing algorithmic thinking skills, *not* about learning a programming language

Algorithms

- Ultimately, algorithms will be implemented by writing code in a particular programming language
- Algorithm design is (mostly) language-independent
  - A procedure that can be implemented in any language
- Universal algorithm representations:
  - Flowcharts
    - Graphical representation
  - Pseudocode
    - Natural language
    - Not necessarily language-independent

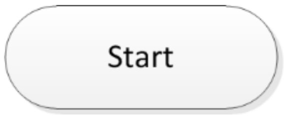
Flowcharts

Flow Charts

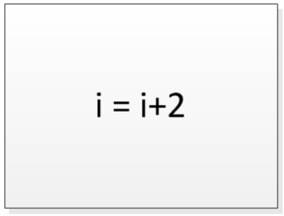
- ***Flowcharts*** are graphical representations of algorithms
- Interconnection of different types of blocks
  - Start/End
  - Process
  - Conditional
  - Input/Output
- Connection paths indicate flow from one step in the procedure to the next
- Well-constructed flowcharts are easily translated into code later

Flowchart Blocks

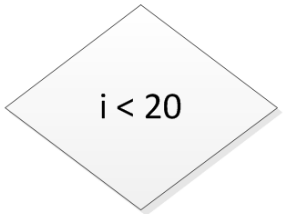
- **Start/End**
  - Always indicate the start and end of any flowchart



- **Process**
  - Indicates the performance of some action



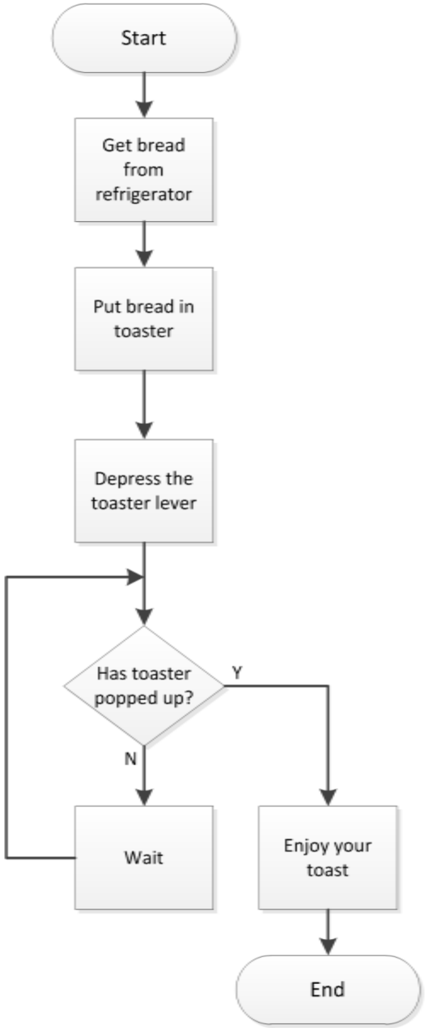
- **Conditional**
  - Performs a check and makes a decision
  - Binary result: True/False, Yes/No, 1/0
  - Algorithm flow branches depending on result



- **Input/Output**
  - Input or output of variables or data

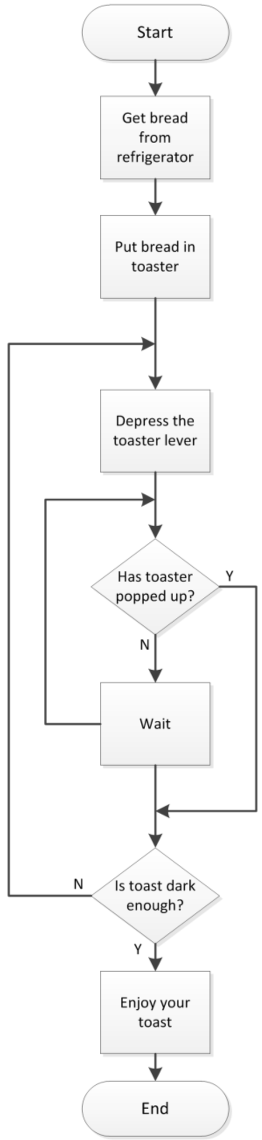
Flowchart – Example

- Consider the very simple example of making toast
- Process flows from Start to the End through the process and conditional blocks
  - Arrows indicate flow
  - Conditional blocks control flow branching
- Note the loop defining the waiting process
  - *Wait* block is unnecessary



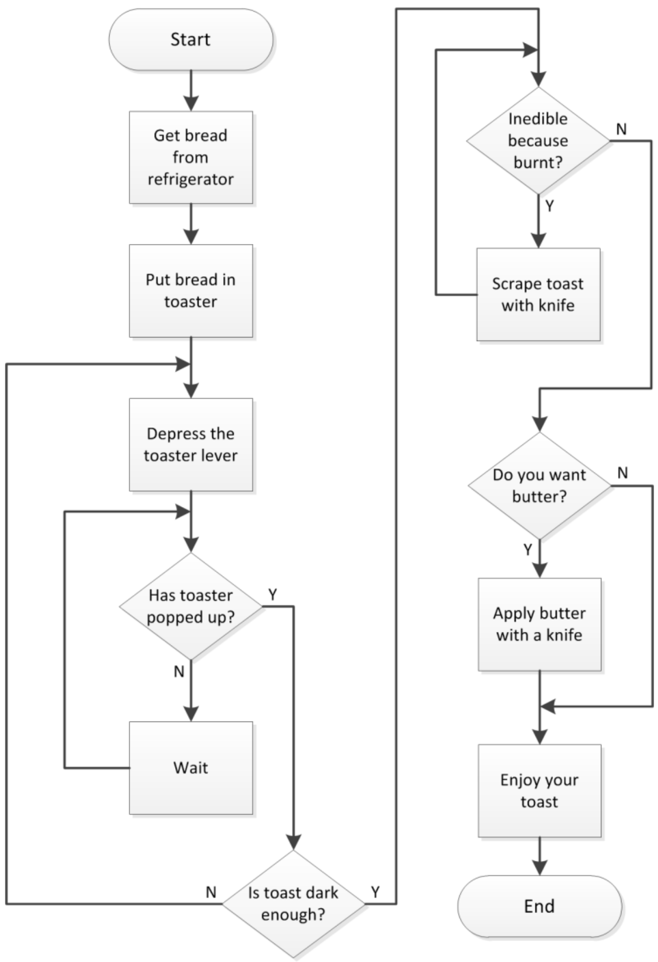
Flowchart – Example

- Flowchart for a given procedure is not unique
  - Varying levels of complexity and detail are always possible
- Often important to think about and account for various possible outcomes and cases
  - For example, is your toast always done after it first pops up?
  - Here, part of the procedure is repeated if necessary



Flowchart – Example

- Taking this example further, consider the possibility of burnt toast or the desire for butter
  - Another loop added for continued scraping until edible
  - Also possible to bypass portions of the procedure – e.g., the scraping of the toast or the application of butter
- Can imagine significantly more complex flow chart for the same simple procedure …



Common Flowchart Structures

Common Flowchart Structures

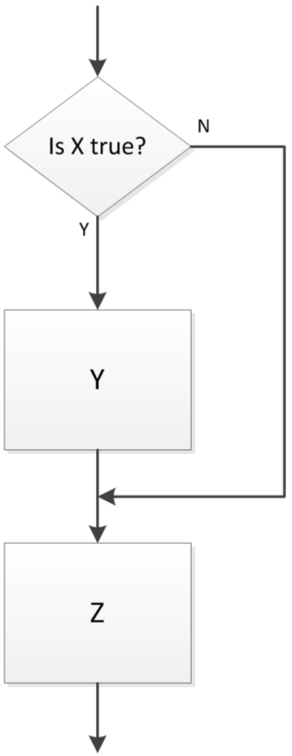
- Several basic structures occur frequently in many different types of flowcharts
  - Recurrent basic structures in many algorithms
- Ultimately translate to recurrent code structures
- Two primary categories
  - ***Conditional statements***
  - ***Loops***
- In this section of notes, we'll gain an understanding of flowchart structures that fall into these two categories
- In the next section of notes we'll learn how to implement these structures in code

Conditional Statements

- `if` statements
- Logical and relational operators
- `if`…`else` statements

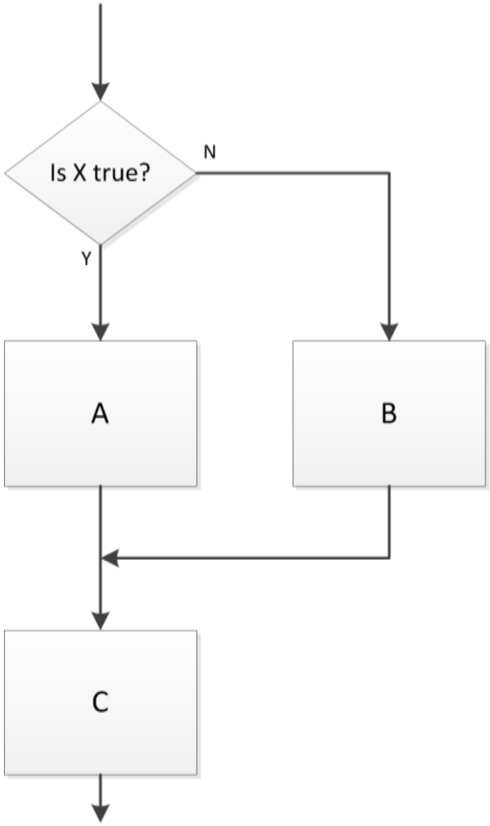
Conditional Statements – *if*

- Flowcharts represent a set of instructions
  - Blocks and block structures can be thought of as ***statements***
- Simplest ***conditional statement*** is a single ***conditional block***
  - An ***if structure***
  - If X is true, then do Y, if not, don't do Y
  - In either case, then proceed to do Z
  - Y and Z could be any type of process or action
    - E.g. add two numbers, turn on a motor, butter the toast, etc.
  - X is a ***logical expression*** or ***Boolean expression***
    - Evaluates to either true (1) or false (0)



Conditional Statements – *if … else*

- Can instead specify an action to perform if X is not true
  - An ***if … else structure***
  - If X is true, then do A, else do B
  - Then, move on to do C
- Here, a different process is performed depending on the value of X (1/0, T/F, Y/N)

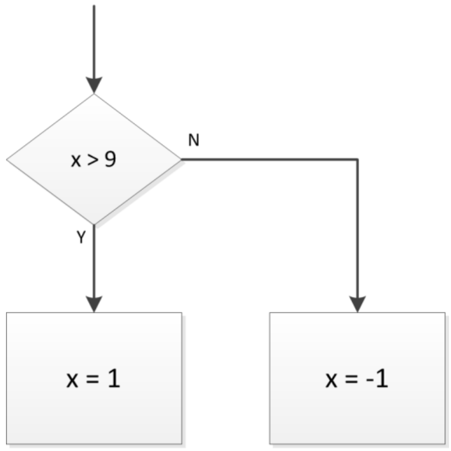


Conditional Statements – *if … else*

- Logical expression with a single ***relational operato***r

$$x > 9$$

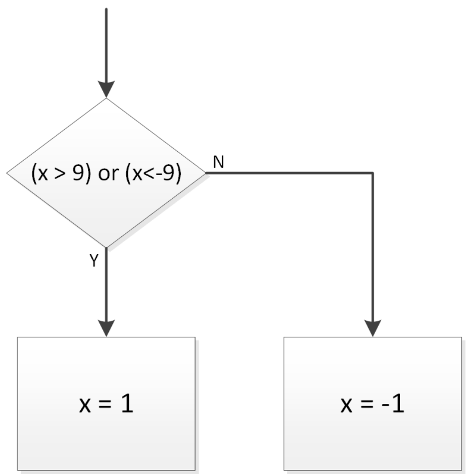
  - Either ***true*** (Y) or ***false*** (N)
  - If true, $x = 1$
  - If false, $x = -1$



- Logical expression may also include a ***logical operator***

$$(x > 9) \text{ or } (x < -9)$$

  - Again, statement is either ***true*** or ***false***
  - Next process step dependent on value of the conditional logical expression



Logical or Relational Expressions

- Logical expressions use ***logical*** and ***relational operators***

Operator	Relationship or Logical Operation	Example
`==`	Equal to	`x == b`
`!=`	Not equal to	`k != 0`
`<`	Less than	`t < 12`
`>`	Greater than	`a > -5`
`<=`	Less than or equal to	`7 <= f`
`>=`	Greater than or equal to	`(4+r/6) >= 2`
`and`	AND – ***both*** expressions must evaluate to true for result to be true	`(t > 0) and (c == 5)`
`or`	OR – ***either*** expression must evaluate to true for result to be true	`(p > 1) or (m > 3)`
`not`	NOT– negates the logical value of an expression	`not (b < 4*g)`

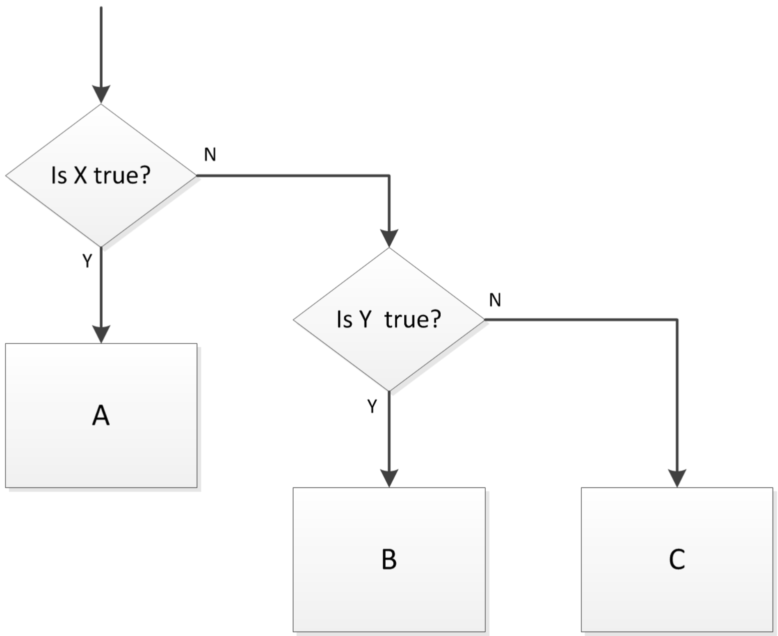
Logical Expressions – Examples

- Let $x = 12$ and $y = -3$
- Consider the following logical expressions:

Logical Expression	Value
$(x + y) == 15$	0
$(y == 2)$ or $(x > 8)$	1
not $(y < 0)$	0
$(y/2 + 1 < -1)$	0
$(x == 12)$ and not $(y \geq 5)$	1
$(y! = 2)$ or $(x < 10)$ or $(x < y)$	1
((x==2) and (y<0)) or ((x≥5) and (y!=8))	1

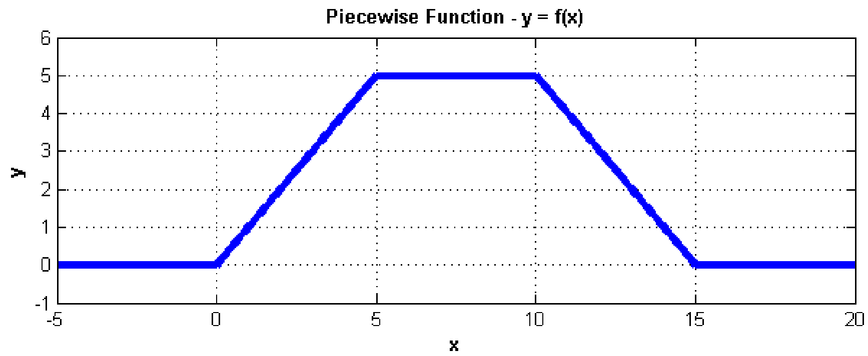
Conditional Statements – *if … elseif … else*

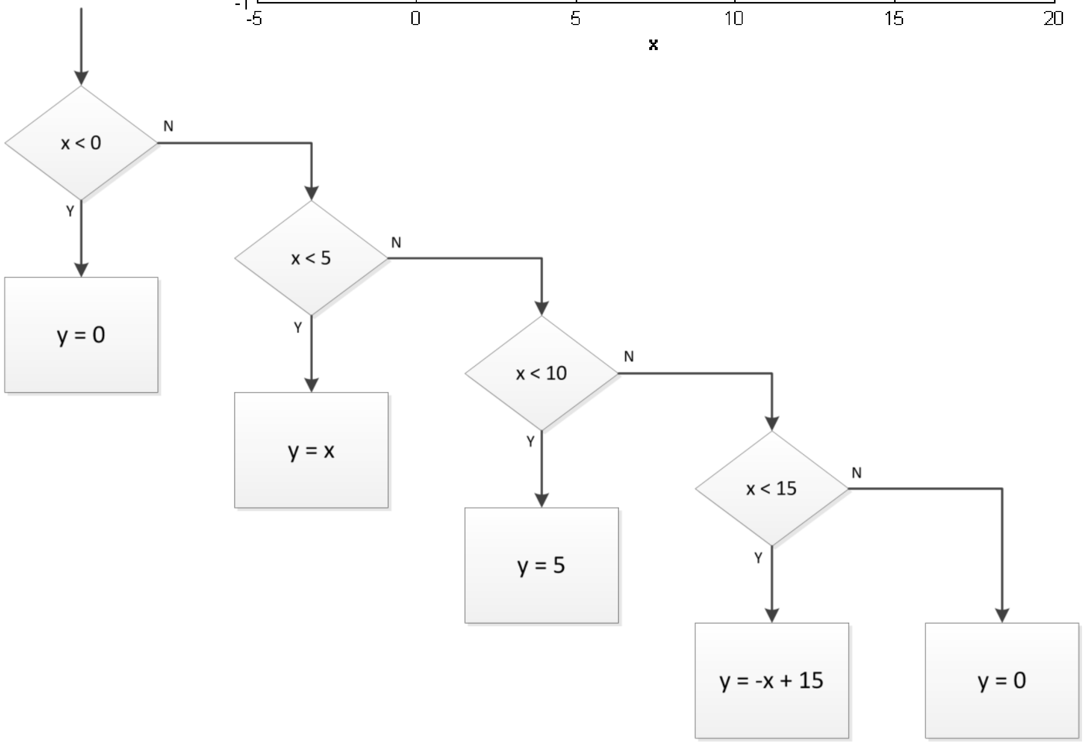
- Two conditional logical expressions
  - If the X is true, do A
  - If X is false, evaluate Y
    - If Y is true, do B
    - If Y is false, do C
- The ***if … elseif … else structure***
- Can include an arbitrary number of *elseif* statements
  - Successive logical statements evaluated only if preceding statement is false



if … elseif … else – Example

- Consider a ***piecewise linear function*** of x
  - $y = f(x)$ not defined by a single function
  - Function depends on the value of x
  - Can implement with an *if … elseif … else* structure





if Statements – Other Configurations

- In previous examples, successive logical statements only evaluated if preceding statement is false
- Result of a true logical expression can also be the evaluation of a second logical expression