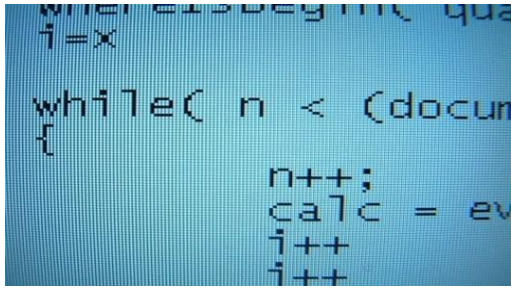
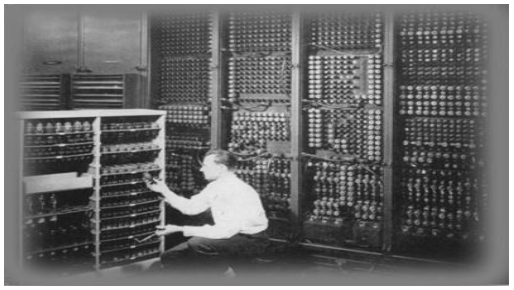




CS1PR16

Algorithmic Thinking

Learning Outcomes

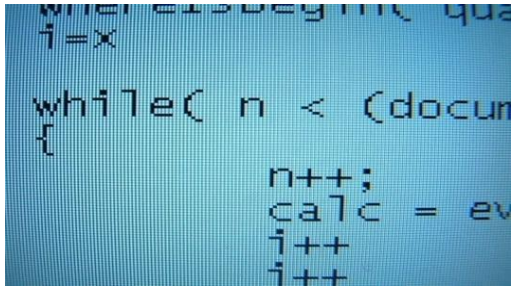
University of Reading

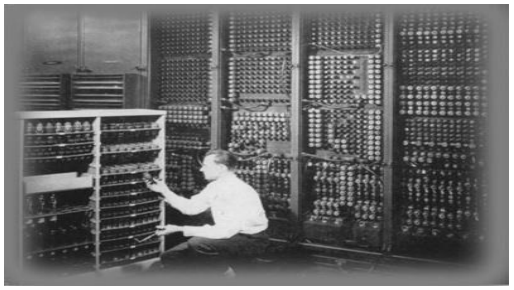
1. Describe the concepts of an algorithm and the Structured Program Theorem and their relation to pseudocode and flow graphs
2. Discuss the quality of short (5-10 line) pseudocode stating two positive or negative aspects
3. Illustrate the control flow graph for a short algorithm that is given as pseudocode
4. Explain the behaviour of a short algorithm (motivated by everyday life activities or mathematical problems) that is provided as pseudocode and outline its behaviour and output by working in small groups.
5. Formulate algorithms (motivated from everyday life activities or mathematical problems) as pseudocode (with minor errors) by working in small groups
6. Describe the relevance of algorithmic thinking and the relation among computer science topics

Outline

University of Reading

- Introduction to Algorithms
- The Structured Programming Theorem
- Describing an Algorithm
- Strategies for Designing Algorithms and Solving Problems
- Programming





Introduction to Algorithms

Definition: Algorithm

University of Reading

*A **procedure** for **solving** a mathematical problem (as of finding the greatest common divisor) **in a finite number of steps** that frequently involves the repetition of an operation*

broadly: a **step-by-step procedure for solving a problem or accomplishing some end**

[Merriam-Webster]

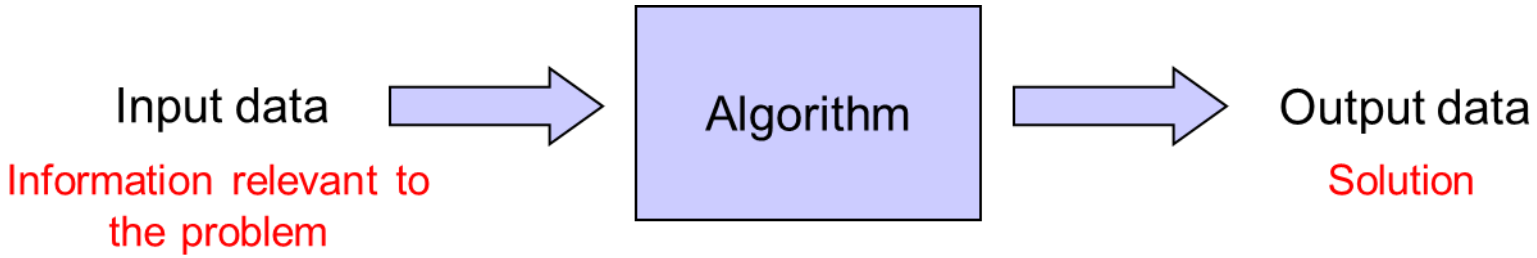
Algorithms are the threads that tie together most of the subfields of computer science.

[Donald Knuth]

Definition: Algorithm: More Precise

University of Reading

An algorithm is a sequence **of unambiguous executable steps, defining a terminating process for solving a problem**



- Input: zero or more inputs, taken from a specified set of objects
- Output: one or more outputs with a specified relation to the inputs

- Properties:
  - **Definiteness**: Each step must be precisely defined. Actions to be carried out must be rigorously and unambiguously specified for each case
  - **Effectiveness**: All operations to be performed must be sufficiently basic that they can be done exactly and in a finite length
  - **Finiteness**: must have an end, thus, indeed produce some kind of output

Question Time

University of Reading

- Can you give an example for an algorithm?
  - Maybe you executed an algorithm?
  - If so, what "algorithm"?
  - What were the inputs/outputs?
- Time: 1 min

Example Algorithms

University of Reading

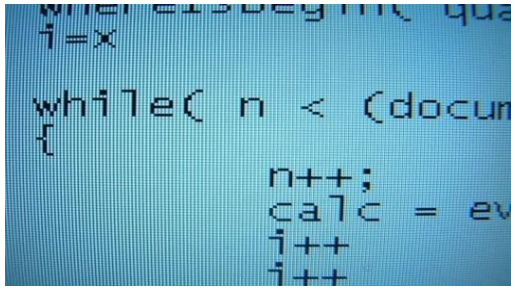
- I'm sure you applied algorithms! They are everywhere!
- Your activities every day
  - Following recipes: How to make a cake
  - Showering, coffee making, driving a car
- Embedded systems
  - The behaviour of an elevator (when pushing a button)
  - Biometric identification on your smartphone
- Math
  - Given two integer values, identify the greater one
  - Find all prime numbers in the interval [1, max], where max is a given value
  - Compute the circumference of a circle or perimeter of a rectangle

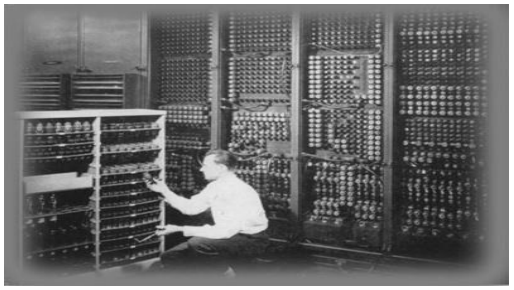
Additional Properties of Algorithms

University of Reading

- **Machine-Independent:** Stays the same independent of
  - Which kind of "hardware" it will execute it (or if manually executed)
  - Which description or programming language it will be written in
- **Correctness**
  - Always returns the desired output for all legal instances of the problem
  - It is always assumed that an algorithm is correct!
- **Efficient**
  - Can be measured in terms of resources
    - Running time (**time complexity**)
    - Memory requirement (**space complexity**)
    - Bandwidth (**communication speed**)
  - For now, only the runtime is relevant

University of Reading





The Structured Programming Theorem

Structured Programming Theorem

(Böhm-Jacopini Theorem; in simplified form)

University of Reading

All algorithms of computable functions can be specified using the constructs:

- **Sequence**: Executing actions consecutively (one, then the next, …)
- **Selection**: Executing one of two actions according to the value of an expression
- **Iteration**: Repeatedly executing actions as long as an expression is true

[Wikipedia]

In this context, actions are composable, i.e., a sequence of actions is a new action => a subprogram

The original theorem uses the term subprogram (instead of action) and applies to control flow graphs. To reduce confusion, the terms have been replaced and simplified

Sequence

University of Reading

- Sequence simply performs one step after another
- Each step is executed in a specific **sequence**
  - do this
  - then do this
  - then do this
- Even better, we can explicitly use numbering
  - Example: (problem: thirsty)
    1. Open the fridge
    2. Take the milk
    3. Drink the milk
- It is the designer of the algorithm who decides about the order
  - In the "milk" example, any different order wouldn't work
  - There are cases where some actions can be reordered without changing the outcome

Selection

University of Reading

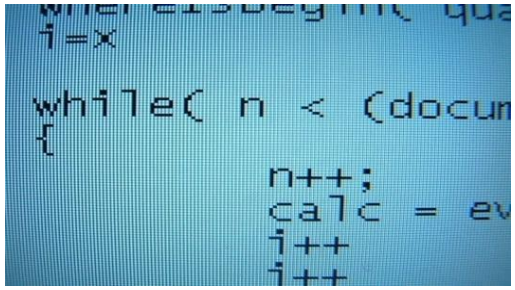
- Selection is the decision-making construct
- It is used to make **yes/no** or **true/false** decisions logically
- Can be considered as:
  - if something is true, then take **this action**
  - otherwise, take **that action**
- Example
  - if it rains, then take a raincoat and an umbrella, else take neither
- Selection is called **conditional statement** in programming languages

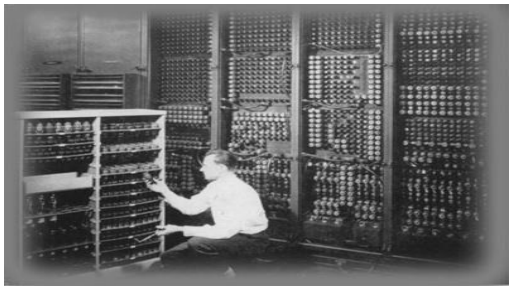
Iteration

University of Reading

- Iteration comes from the word reiterate, which means to repeat
- Iteration is a **looping** construct
- Iteration repeats a sequence of actions based on a decision
- Explicit number of iterations
  - repeat this five times
  - or repeat this X times, where X is the age of a person
- Implicit number of iterations
  - While something is **true**, keep **doing this,** otherwise stop

University of Reading





Describing an Algorithm

Describing an Algorithm

University of Reading

- Thanks to the structured programming theorem, we can think of how to describe algorithms
  - Notations are important to share/understand algorithms
- There are various ways of expressing an algorithm:
  - Textual:
    - Natural language
    - Pseudocode
    - Programming language
  - Graphical
    - Flowcharts
- Quality of representations differ
  - All must share: definiteness, effectiveness, finiteness
  - **Understandability and standards are important**

Natural Language

University of Reading

Recipes: More or less standardised description

Hummus and Tomato Pasta
Serves 2
This is a very quick 20-minute after work dish.

Ingredients:
Olive oil
1 teaspoon of whole cumin seeds
1 large chopped onion
400g chopped plum tomatoes
200g hummus
150g pasta

Steps:
1. Add the pasta to a large pan of boiling water. Simmer for 10 minutes.
2. Fry the cumin in the olive oil for two minutes. Add the onions and fry gently.
3. Stir in the tomatoes and the hummus and leave to simmer until done.
4. Serve the pasta and add the sauce.

Based upon http://www.cs4fn.org/programming/recipeprogramming.php

Pseudocode

University of Reading

- Pseudo means "pretend" or "false"
- Pseudocode pretends to be computer code
  - the generic formulation that is *somewhat like* computer code
  - "somewhat like" because it uses **constructs** of programming languages
- Pseudocode syntax is not standardised
  - It is a semi-formal representation of an algorithm
  - There are various forms of pseudocode
    - Some are closer to a programming language
    - Some are closer to natural language

Real World Pseudocode Example

University of Reading

Pseudocode: How to Start a Car (two variants with slight changes)

Note that some actions in prose are a bit vague

```
1.   Insert Key in Ignition
2.   Turn Key to Start Position
3.
4.   Repeat while Engine Hasn't Started
5.
6.       Hold Key In Start Position
7.
8.       If 6 Seconds have Passed then
9.           Turn Key to Off Position
10.          Wait 10 Seconds
11.          Turn Key to On Position
12.      End If
13.
14.  End Repeat
```

```
Insert Key in Ignition
Turn Key to Start Position

while Engine Hasn't Started do
    Hold Key in Start Position

    if 6 Seconds have Passed then
        Turn Key to Off Position
        Wait 10 Seconds
        Turn Key to On Position
    end

end
```

Another “Algorithm“ / PseudoCode

University of Reading

Hummus and Tomato Pasta
Serves 2
This is a very quick 20-minute after work dish.

Ingredients:
Olive oil
1 teaspoon of whole cumin seeds
1 large chopped onion
400g chopped plum tomatoes
200g hummus
150g pasta

Steps:
1. Add the pasta to a large pan of boiling water. Simmer for 10 minutes.
2. Fry the cumin in the olive oil for two minutes. Add the onions and fry gently.
3. Stir in the tomatoes and the hummus and leave to simmer until done.
4. Serve the pasta and add the sauce.

Pseudocode: Close to Programming

University of Reading

```
RECIPE Hummus and Tomato Pasta ()
{
/* Serves 2
This is a very quick after work dish.
It only takes about 20 minutes from start to finish.
*/

  /* Define tools to use */
  Saucepan pan1;
  Fryingpan pan2;
  Kettle       kettle;

  /* Define the Ingredients as variables */
  Dish oilDish = 1 tablespoon of olive oil;
  Dish cuminDish = 1 teaspoon of whole cumin seeds;
  Dish onionDish = 1 large onion, chopped;
  Dish tomatoDish = 400g chopped plum tomatoes;
  Dish hummusDish = 200g hummus;
  Dish pastaDish = 150g pasta;
  Dish water = 500ml;

  /* Cook the pasta */
  kettle = water;
  Heat(kettle, boil);
  pan1 = kettle + pastaDish;
  Heat (pan1, high, 2 minutes);
  Heat (pan1, medium, 10 minutes);

  /* Make the sauce */
  pan2 = oilDish + cuminDish;
  Heat (pan2, high, 2 minutes);
  pan2 = pan2 + onionDish;
  Heat (pan2, medium, 5 minutes);
  pan2 = pan2 + tomatoDish + hummusDish;
  Heat (pan2, low, 5 minutes);

  /* serve */
  /* pour 50% remaining in the pans */
  Plate p1 = Pour(pan2, 50%) + Pour(pan1, 50%)
  /* pour 100% remaining in the pans */
  Plate p2 = Pour(pan2, 100%) + Pour(pan1, 100%)
}
```

Group Work: General Introduction

University of Reading

- Now we will do a short group work!
  - Following the "Think-Pair-Share" strategy
- This part of the lecture works as follows:
  - First, **think** and **discuss** with your direct neighbour (2-3 people)
  - Then, we share and discuss in the whole class
    - The lecturer will choose people to share their thoughts
    - You can raise your hand ☺
- We will have a time limit

Group Work 1

- Question: Is this a "good" description for an algorithm?
  - **Think if they meet the requirements**
    - **Definiteness:** is each step precisely defined?
    - **Effectiveness:** is each step basic enough and can be executed in practice?
    - **Finiteness:** does it terminate?
  - **How "understandable" is the algorithm for YOU? Can you execute it?**
  - Start to think about: How would you define understandability?
- Time: 3 min
- Share: 3 min

Hummus and Tomato Pasta
Serves 2
This is a very quick 20-minute after work dish.

Ingredients:
Olive oil
1 teaspoon of whole cumin seeds
1 large chopped onion
400g chopped plum tomatoes
200g hummus
150g pasta

Steps:
1. Add the pasta to a large pan of boiling water. Simmer for 10 minutes.
2. Fry the cumin in the olive oil for two minutes. Add the onions and fry gently.
3. Stir in the tomatoes and the hummus and leave to simmer until done.
4. Serve the pasta and add the sauce.

Group Work 1: A Bit Unclear?

University of Reading

Hummus and Tomato Pasta
Serves 2 **[Useful to know, kind of the output]**
This is a very quick 20-minute after work dish. **[Describes the "performance"]**
Ingredients: [Useful to know, kind of the input]
Olive oil
1 teaspoon of whole cumin seeds
1 large chopped onion
400g chopped plum tomatoes
200g hummus
150g pasta

Steps:
1. Add the pasta to a large pan [ok, large to cover all pasta] of boiling water. Simmer for 10 minutes.
2. Fry the cumin in the olive oil [in another pan?] for two minutes. Add the onions and fry gently.
3. Stir in the tomatoes and the hummus [to which pan? The sauce?] and leave to simmer until done [How do I know when it is done?].
4. Serve the pasta and add the sauce [The sauce is in the second pan?].

Definitness? Understandability?

University of Reading

- This video shows "literal" execution of instructions for making a Peanut-Butter-Jelly sandwich
- https://youtu.be/cDA3_5982h8?t=19
- Thus, it can be hard to write down algorithms!

Group Work 2

University of Reading

- Question: Is this a "good" description for an algorithm?
  - **Think if they meet the requirements**
    - **Definiteness:** is each step precisely defined?
    - **Effectiveness:** is each step basic enough and can be executed in practice?
    - **Finiteness:** does it terminate?
  - **How "understandable" is the algorithm for YOU? Can you execute it?**
  - **Now: How would you define understandability?**
- Time: 2 min
- Share: 2 min

```
Insert Key in Ignition
Turn Key to Start Position

while Engine Hasn't Started do
    Hold Key in Start Position

    if 6 Seconds have Passed then
        Turn Key to Off Position
        Wait 10 Seconds
        Turn Key to On Position
    end

end
```

Group Work 2

University of Reading

- **Firstly, this "algorithm" does not necessarily terminate**
  - Imagine the engine is broken
- **Why is this a problem?**
  - What would happen in real life if your "problem" to solve is to start a **broken car** and you apply this algorithm?
  - Do you understand why Finiteness is an important property to solve problems?
- What would you change to make it work?
  - Does it become more complicated?
- Would it been useful to add a description:
  - Problem: "Making a car ready to drive"?
  - Input: Key, Car
  - Output: Ignited car
- Otherwise, it appears easy to understand!

```
Insert Key in Ignition
Turn Key to Start Position

while Engine Hasn't Started do
    Hold Key in Start Position

    if 6 Seconds have Passed then
        Turn Key to Off Position
        Wait 10 Seconds
        Turn Key to On Position
    end

end
```

Group Work 2

University of Reading

- **Other problems:**
  - **Hold key in start position vs. Turn key to on position?**
  - **The condition is always true (after 6 seconds)**

- **What happens actually?**
  - **Turn on, sleep, turn off, wait, turn on**
    - **Then check immediately if the car is on**
    - **We just moved the key, so likely it is off...**

- **How can we fix the algorithm?**
  - **Ideas?**

```
Insert Key in Ignition
Turn Key to Start Position

while Engine Hasn't Started do
    Hold Key in Start Position

    if 6 Seconds have Passed then
        Turn Key to Off Position
        Wait 10 Seconds
        Turn Key to On Position
    end

end
```

Algorithm How to Start a Car

University of Reading

Algorithm StartCar:

let tries = 0

while *car is off* **do**

    tries = tries + 1

    turn key to on position

    sleep 2s

    if *car is off* **and** *tries > 3* ***then***

        print "it is dead, Jim. Please call a mechanic."

        exit

    end

end

```
Insert Key in Ignition
Turn Key to Start Position

while Engine Hasn't Started do
    Hold Key in Start Position

    if 6 Seconds have Passed then
        Turn Key to Off Position
        Wait 10 Seconds
        Turn Key to On Position
    end

end
```

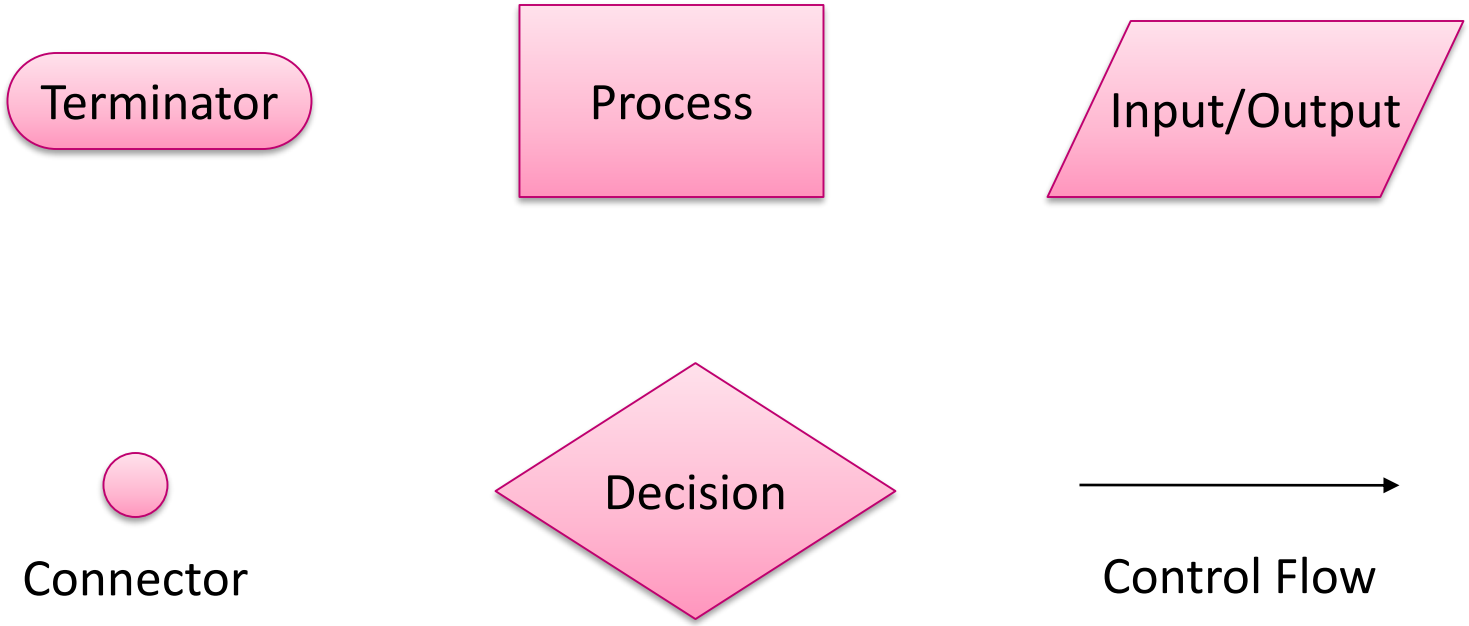
Definiteness: Design a Good Description

University of Reading

- Documenting an algorithm serves the purpose to communicate
  - Also allows to "implement" the algorithm such that it can be "executed"
- A documented algorithm should be:
  - Analysable: *The ability to be analysed*
  - Understandable for the target audience (CS students?)
  - **Have an** "unambiguous description" (**Definiteness!)**
- How to design a good description of an algorithm:
  - Define the terms used
  - Clarify Purpose/Input/Output
  - Add natural language to describe in prose how the algorithm works
  - Be consistent with terms for **sequence, iteration, selection of steps**
  - For each step, ensure that it is clear what it does and how it works
  - Make sure that the algorithm works for **all possible inputs**
  - Handle errors!

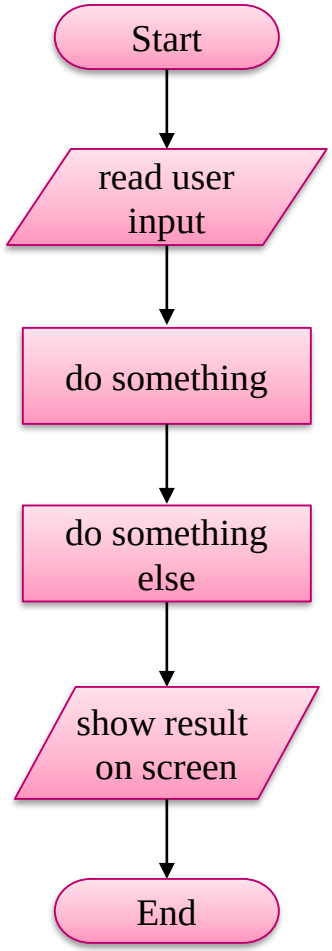
Flowcharts

- A Flowchart is a visual representation of an algorithm
  - Alternative to pseudocode, serves the purpose of communication
- Flowcharts are well-defined and standardised
  - Use easy-to-understand symbols to represent steps
  - Visual elements represent the constructs of programming languages
- Symbols:



Flowchart for Sequence

Flowchart:

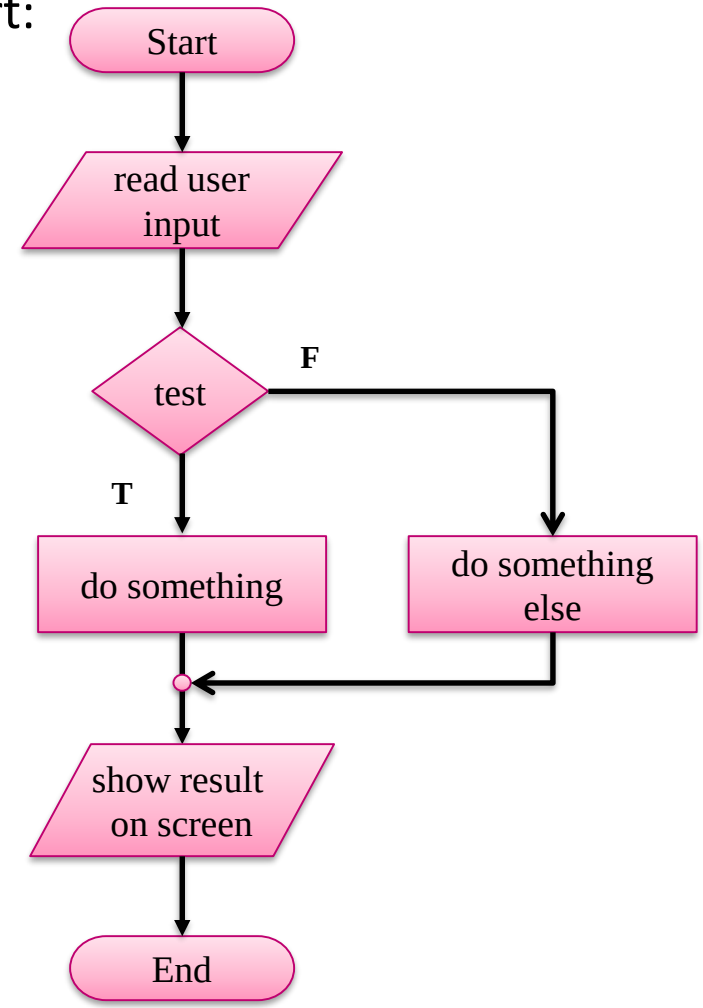


Pseudocode:

1. Read User Input
2. Do Something
3. Do Something Else
4. Show Result on Screen

Flowchart for Selection

Flowchart:

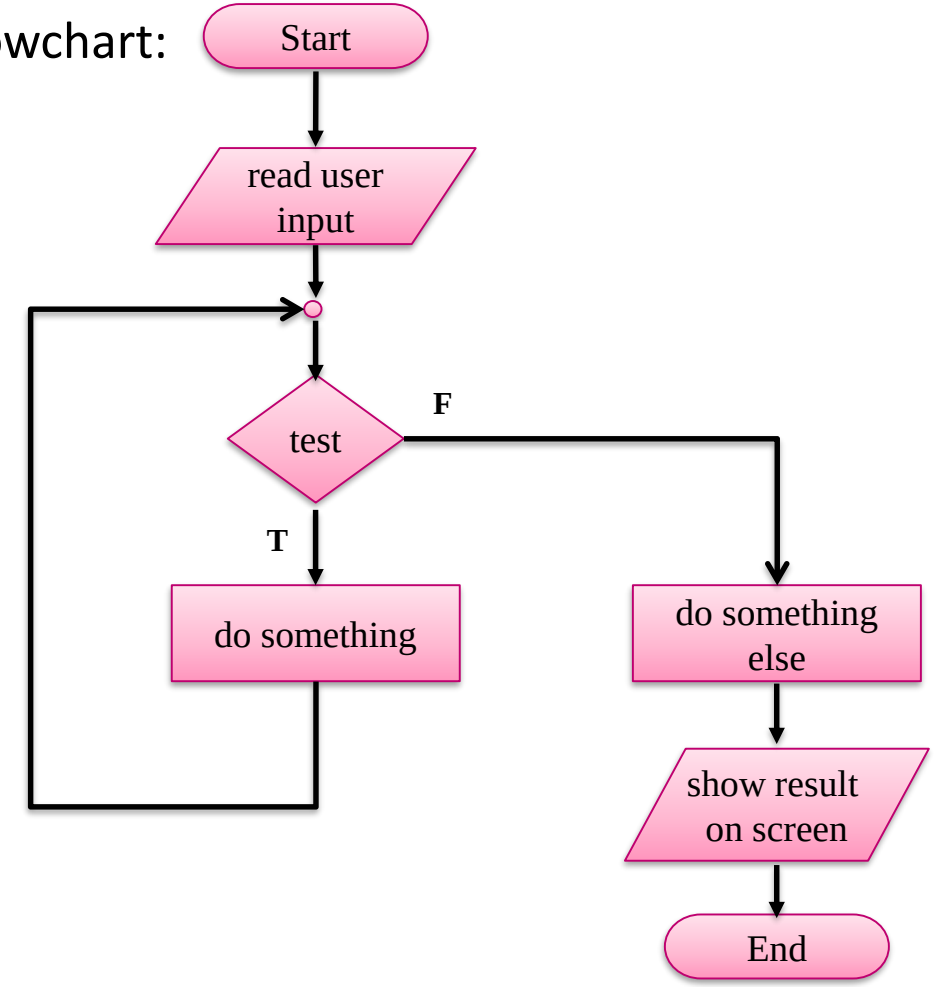


Pseudocode:

```
1.  Read User Input
2.
3.  If the test is satisfied (True)
4.      Do Something
5.  Else (the test is not satisfied (False))
6.      Do Something Else
7.  End If
8.
9.  Show Result on Screen
```

Flowchart for Iteration

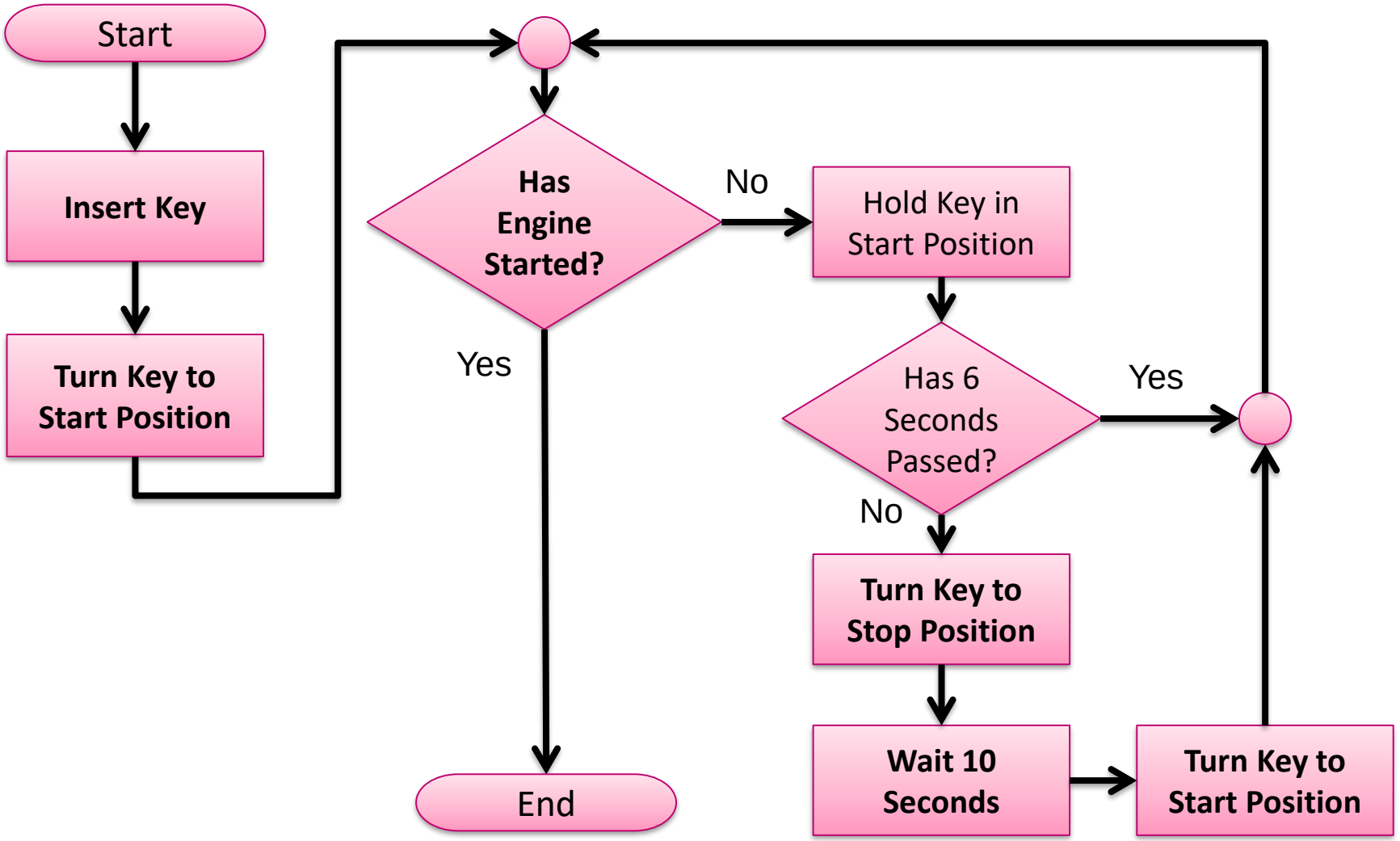
Flowchart:



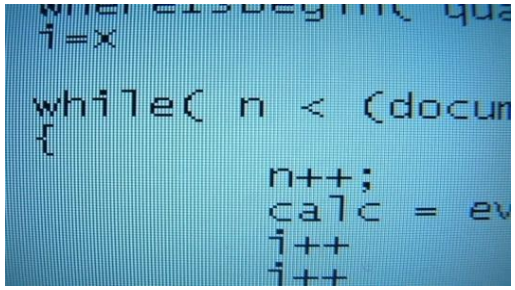
Pseudocode:

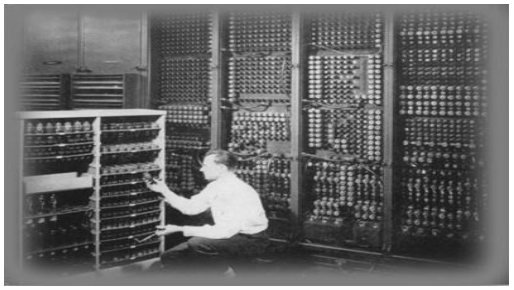
```
1.  Read User Input
2.
3.  While test is true
4.      Do Something
5.  End While
6.
7.  Do Something Else
8.  Show Result on Screen
```

Flowchart Examples: Start a Car



University of Reading





Strategies for Designing Algorithms and Solving Problems

Introduction to Problem-Solving

University of Reading

Problem-solving: Design an algorithm to solve a problem

An informal taster of strategies:

- Ask questions
- Look for similar problems that are familiar
- Means-Ends analysis
- Divide and Conquer

The strategies can be combined

Strategy: Ask Questions

University of Reading

- **Start with a high-level description** of the algorithm, **including** questions in the description and **refining** the description until resolved
  - What, Why, When, and Where?
  - What do you know?
  - What do you not know?
- In the context of problem-solving
  - What is my input (What do I have to work with)?
    - What do the data items look like?
    - How much data are there?
  - What should my output look like?
  - How will I know when I have processed all the data?
  - How many times is the process going to be repeated?
  - What **error conditions** might come up?

Strategy: Look for Familiar Things

University of Reading

- Never reinvent the wheel
  - If a solution for a similar problem exists… use it, or abstract it
- Finding the daily high and low temperatures
  - is the same problem as finding the highest and lowest grades on a test
- Both problems can be abstracted as
  - find the largest and smallest values in a set of numbers

Strategy: Means-Ends Analysis

University of Reading

- Beginning state (Input) and end state (Output) are often given
- Define a set of actions to **get from start to end**
- Once you have a set of actions, you need to work out the details

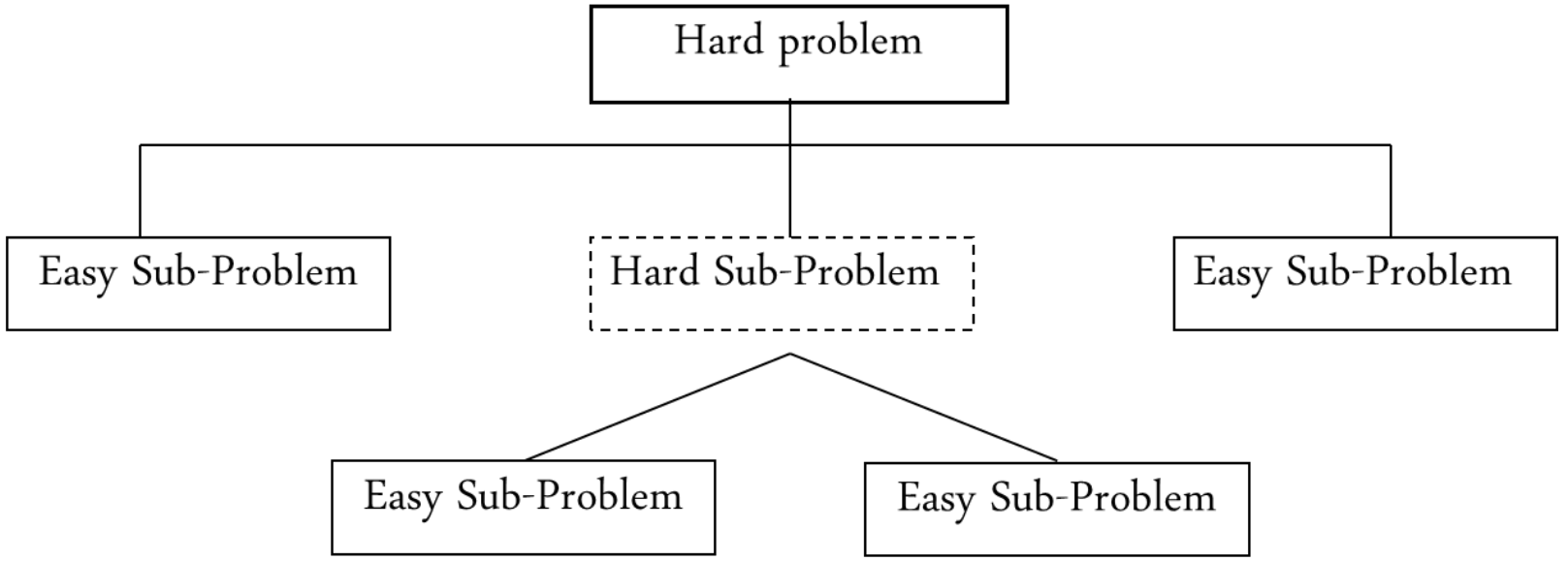
- Translated to algorithm development
  - Begin by writing down: what is the input? (Beginning state)
  - What should the output be? (End state)
  - What actions can be performed to obtain results from the input data?

Strategy: Divide and Conquer

University of Reading

Decomposition of problems into smaller ones that are easier to handle

Top-Down approach:



Example: Area of a Circle

University of Reading

Algorithm: Compute the area of a circle

Problem statement

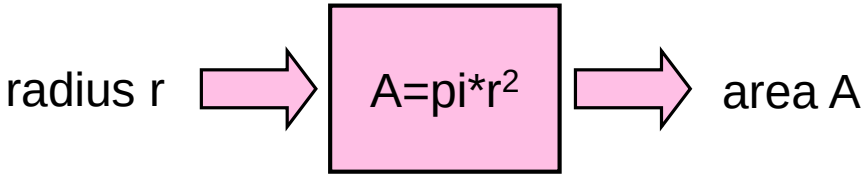
We need an interactive program (user will input data) that computes the area of a circle. Given the circle radius, the circle area should be displayed on the screen.

Input/Output description

Input → Circle radius

Output → Circle area

Graphical representation of the processing with in/output



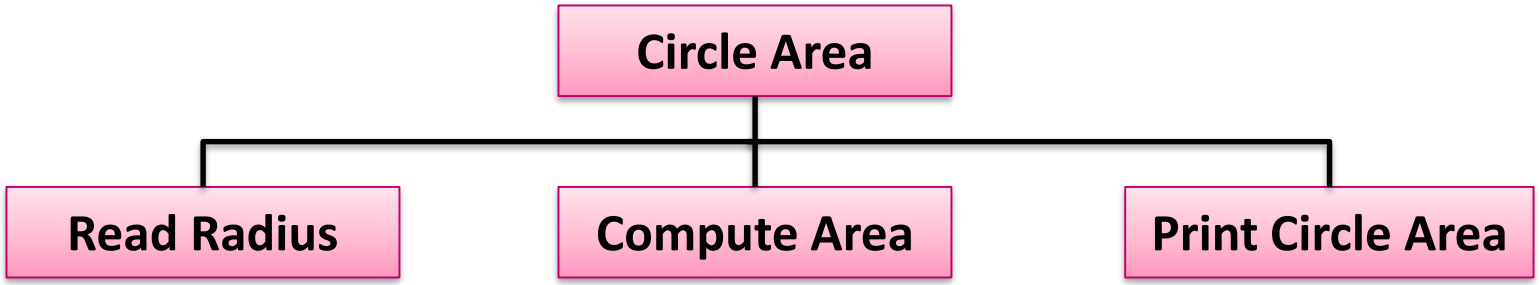
Algorithm in Natural Language (high-level set of steps, decomposition outline)

1. Read value of circle radius (r) from the user
2. Compute circle area as pi*r*r
3. Print the value of circle area on the screen

Example: Area of a Circle

University of Reading

A divide and conquer block diagram of our problem



Do we understand each of the "subproblems" to solve them?

Example: Area of a Circle

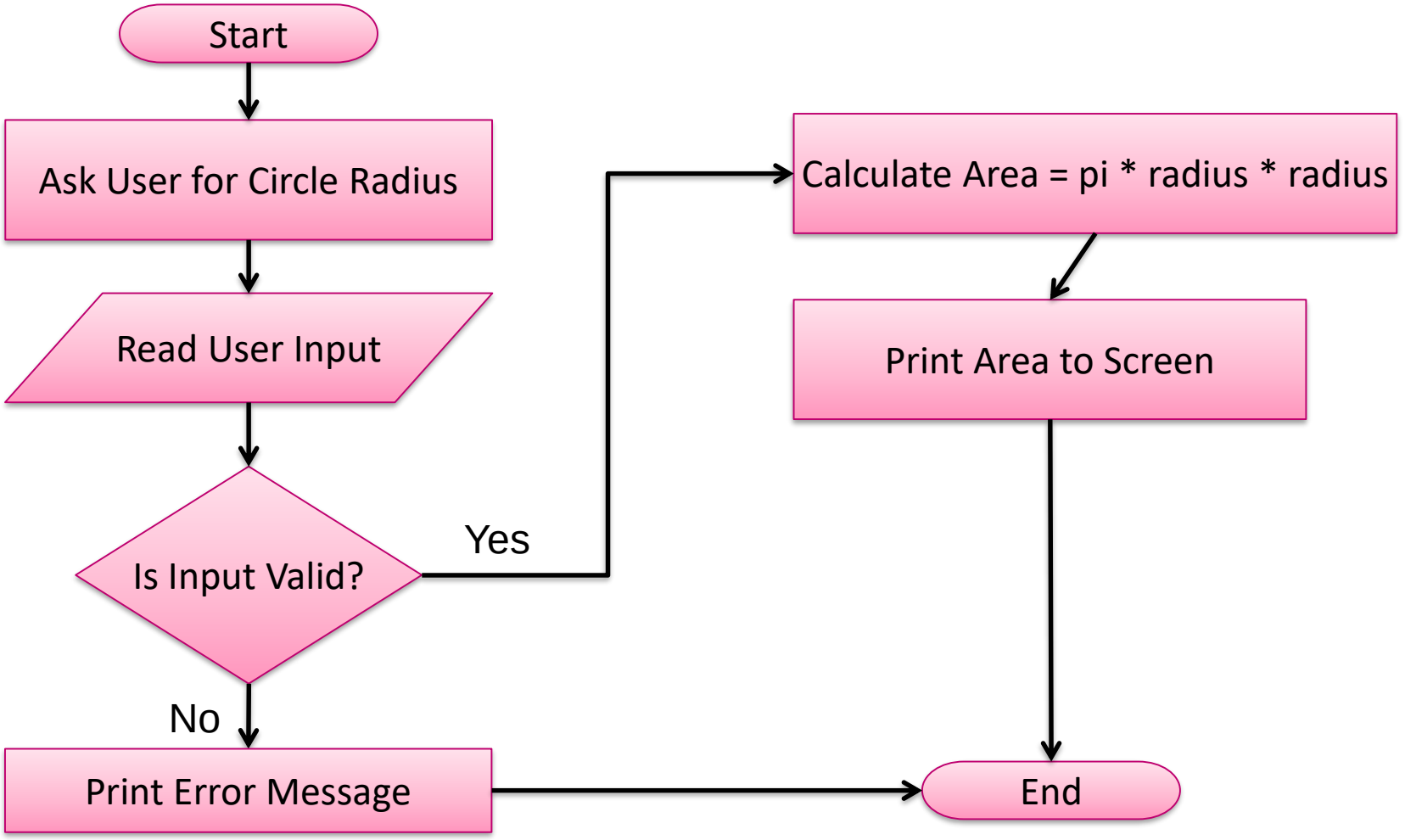
University of Reading

Pseudocode

1. **Prompt** the user for the circle radius (write a message on the screen)
2. **Read** the value of the "radius" as provided by the user
3. If the value is not valid (e.g., a negative number) print an error message on the screen and exit returning an error code.
4. **Assign** circle "area" **the value** pi * radius*radius
5. **Write** the value of the circle "area" on the screen, return with the success code

Details of the error handling are out of scope for now…

Flowchart: Area of a Circle



Group Work 3

University of Reading

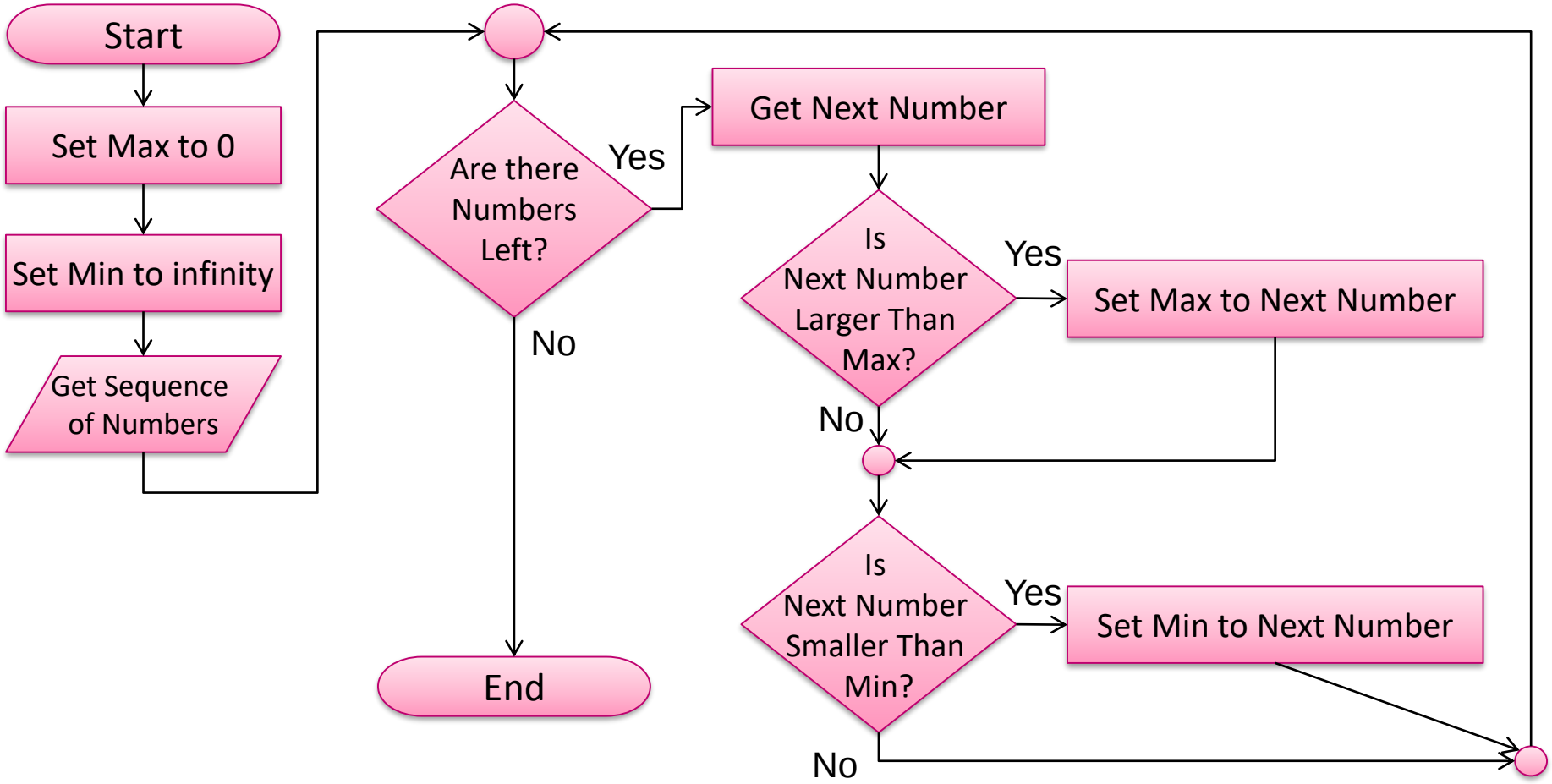
Sketch an algorithm with your neighbour

- Choose one algorithm:
  - EASY: how to make coffee?
  - HARDER: how to determine the range of a set of numbers
    - e.g. find the minimum and maximum value!
- Time: 3 min
- Share: 3 min

Range Algorithm Example

University of Reading

Flowchart: Range



Range Algorithm Example

University of Reading

Pseudocode: Range

```
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
```

Set Max to 0
Set Min to the highest number possible
Get Sequence of Numbers

While there are numbers left in the sequence

    Get to the next number in sequence (first number if first time)

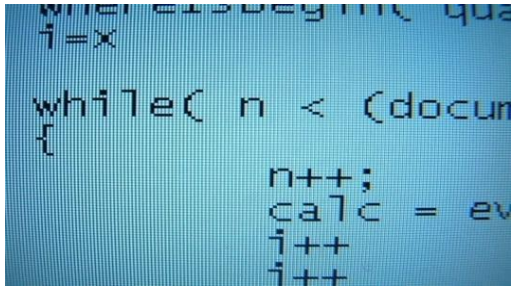
    If the number is greater than Max
        Set Max to current number
    End If

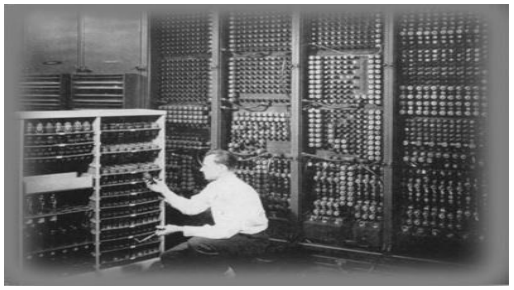
    If the number is smaller than Min
        Set Min to Current Number
    End If

End While

Set Range to (Min, Max)
Output Range

University of Reading





Programming

Definition: Programming

University of Reading

Computer programming *is the process of* ***designing*** *and* ***building*** *an executable computer program for accomplishing a specific* ***computing*** ***task****.*

Programming *involves tasks such as:*

- ***analysis, generating algorithms, profiling algorithms' accuracy and resource consumption****, and*
- *the* ***implementation*** *of algorithms in a chosen programming language (commonly referred to as* ***coding)***

The ***purpose*** *of* ***programming*** *is to* ***find a sequence of instructions*** *that will* ***automate the performance of a task*** *(which can be as complex as an operating system) on a computer,* ***often for solving a given problem****.*

The process of programming thus often requires expertise in several different subjects, including knowledge of the application domain, specialised ***algorithms****, and formal logic.*

[Wikipedia]

Team effort: More complex programs are programmed in teams

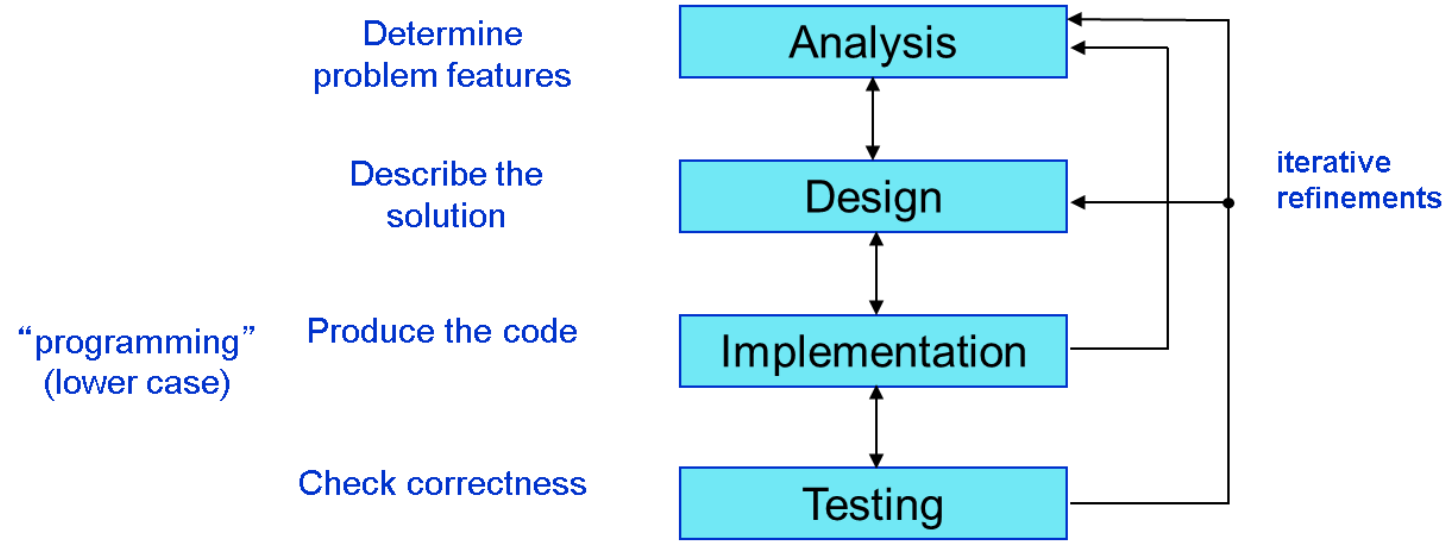
A ***programming language*** *is a* ***vocabulary*** *and set of* ***grammatical rules*** *for instructing a computer or computing device to* ***perform specific tasks***

- **Kind of a standardised form to describe algorithms that can be executed!**

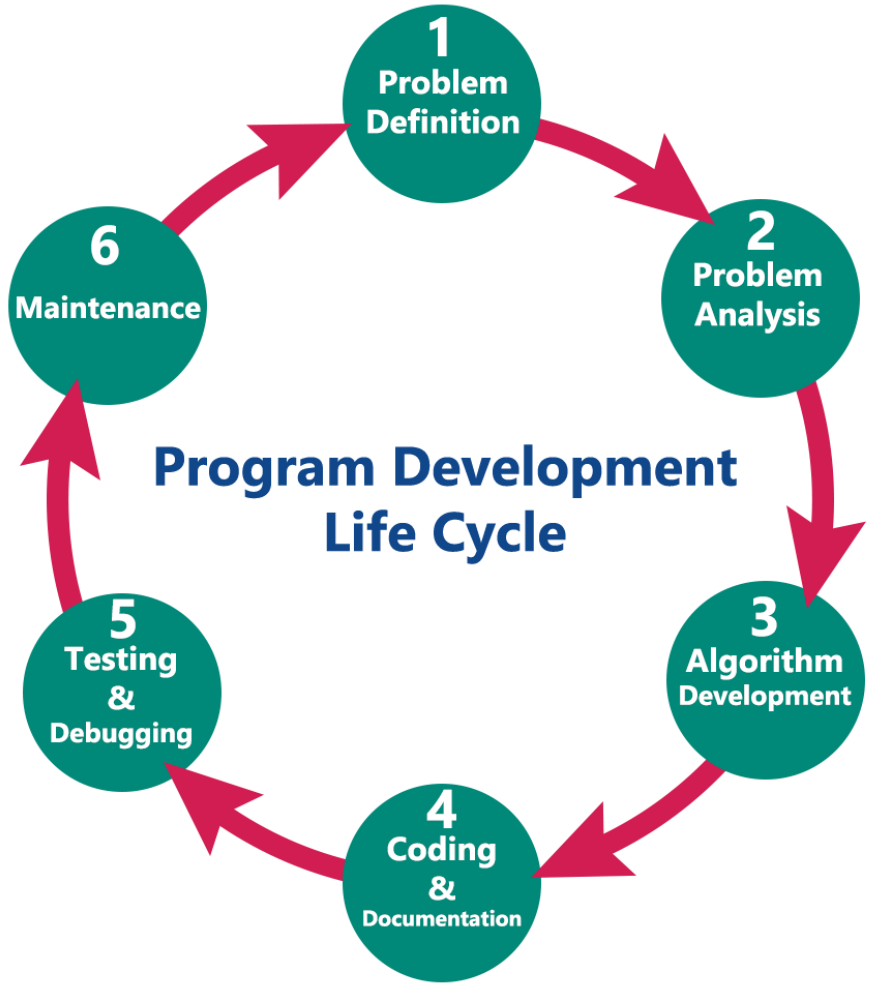
Programming

Programming is a process of problem-solving

- Expanding upon algorithmic thinking
- Write an algorithm and encode (implement) it in a program
- Make it such that the computer can "execute" the algorithm



Program Development Lifecycle



Source: http://btechsmartclass.com/CP/program-development.htm
Also see: https://www.youtube.com/watch?v=YV6ykfG1nQY

This cycle can be seen as a strategy on how to solve any problem – **in your daily life!**

Today we focused on
2) Problem analysis
3) Algorithmic development

We also talked a bit about:
1) problem definition

Programming as Problem-Solving

University of Reading

- The purpose of writing a program is to solve a problem
- The general steps for programming are:
  - Understand the problem
  - Dissect the problem into manageable pieces
  - Design an algorithm for the pieces
    - Consider alternatives to the solution and refine it
    - Analyse the complexity of the algorithm
  - Implement the solution
  - Test the solution and fix any problems that exist ☺
    - This process is commonly referred to as "**debugging**"

Wider Context of Programming

University of Reading

Tasks accompanying and related to programming include:

- testing, debugging, source code maintenance,
- implementation of build systems, and management of derived artefacts, such as the machine code of computer programs.

These might be considered part of the programming process, but often the term

- *software development* is used for this larger process with the term
- ***programming*, *implementation*, or *coding* reserved for the actual writing of code**.
- *Software engineering* combines engineering techniques with software development practices.

A ***hacker*** is any skilled computer expert that uses their technical knowledge to overcome a problem, but it can also mean a *security hacker* in common language.

[Wikipedia]

Based on this definition, you will become a **hacker** as part of this course!

Software Development & Engineering

University of Reading

- Most software projects fail because
  - the developer didn't really understand the **requirements of the problem**
  - **communication in teams is not effective (the human factor)**
- As problems and their solutions become larger, we must
  - Apply software engineering strategies to manage our coding AND teams
  - Organise our development into manageable pieces
  - Provide documentation to avoid assumptions and clarify ambiguities
- These techniques are fundamental to software development

Summary

University of Reading

- Every algorithm can be coded using:
  - Sequence, Selection, Iteration
- Algorithm:
  - ***unambiguous executable steps, defining a terminating process for solving a problem***
  - **Mandatory requirements: Definiteness, Effectiveness, Finiteness**
  - **Properties: Performance, Correctness, Machine-Independent**
- Describing Algorithms:
  - Natural language, pseudocode, flowcharts
- Designing Algorithms:
  - Divide and Conquer strategy; always define Input/Output
- Next week: (Computer) System Architectures