

Programming Techniques 1-(JAVA) BCNS1102C

**Lecture 1_2: Introduction:
Problem Solving & Algorithm
Lecturer: Mr. Ajit Kumar Gopee**

E-mail: ajit.gopee@utm.ac.mu

Outline

- Hardware & Software
- Programming Languages
 - Machine Language
 - Assembly Language
 - High Level Language
 - Compiler & Interpreter
 - Java
- Problem Solving
 - Algorithms
 - Flowcharts
 - Pseudocode

- Hardware
 - the physical, tangible parts of a computer
 - keyboard, monitor, disks, wires, chips, etc.
- Software
 - programs and data
 - a *program* is a series of instructions
- A computer requires both hardware and software
- Each is essentially useless without the other

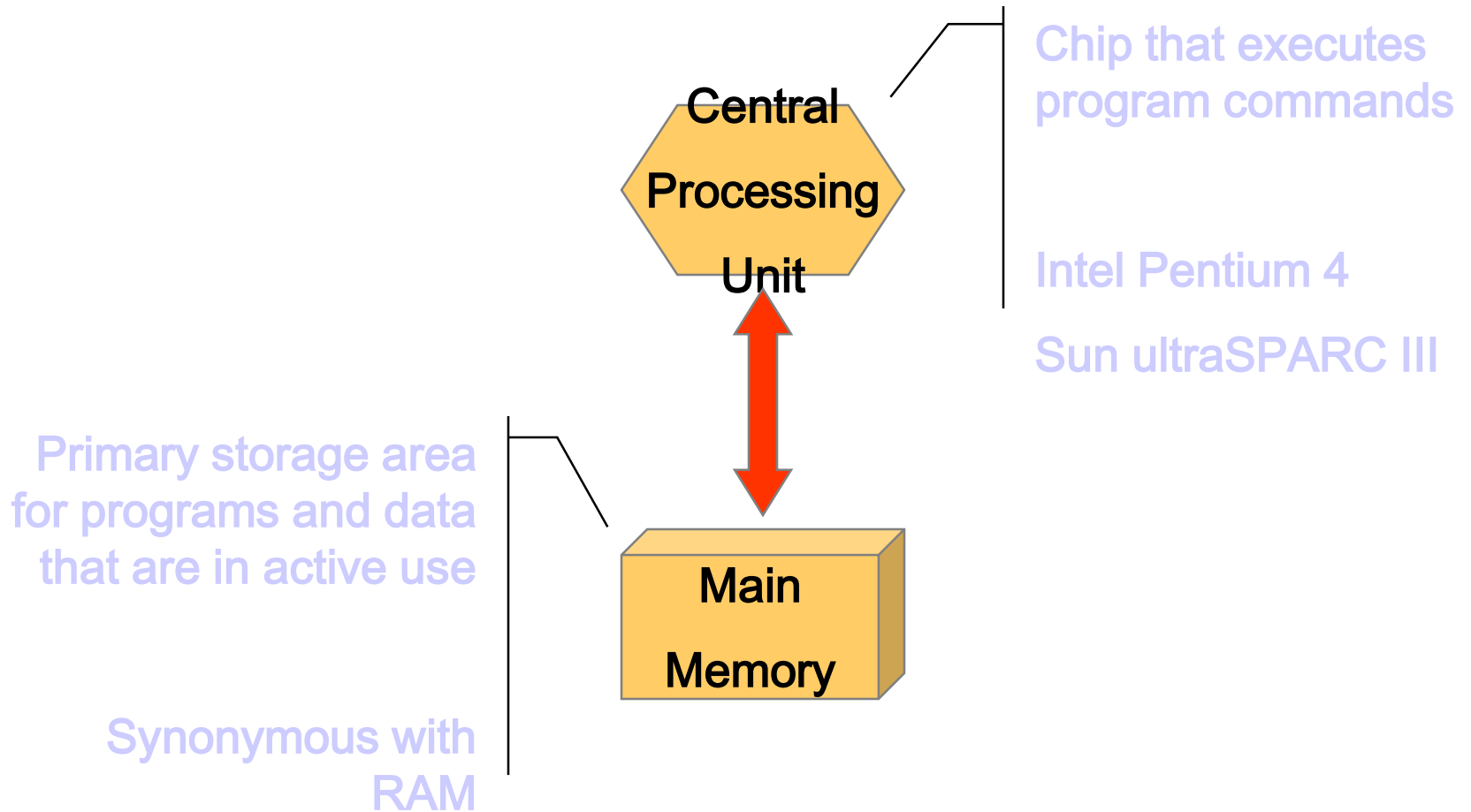
Computers and Computer Languages

- Computers are everywhere
 - how many computers do you own?
- Computers are useful because they run various programs
 - program is simply a set of instructions to complete some task
 - how many different programs do you use in a day?

Introduction

- A computer is a complex system consisting of various components
 - Central Processing Unit (CPU)
 - Main Memory (RAM)
 - Secondary Storage
 - I/O sub-system
- Main role of the CPU, is to execute programs.
- A Program is a list of instructions which are followed/executed by the CPU.
 - CPU understands machine language
 - Each type of computer has its own machine language

CPU and Main Memory



Secondary Memory Devices

Secondary memory devices provide long-term storage

Hard disks

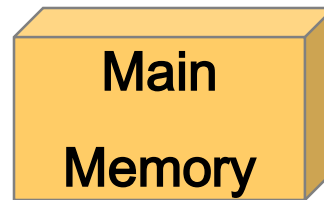
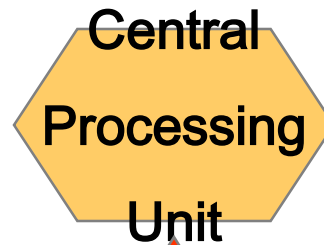
Floppy disks

ZIP disks

Writable CDs

Writable DVDs

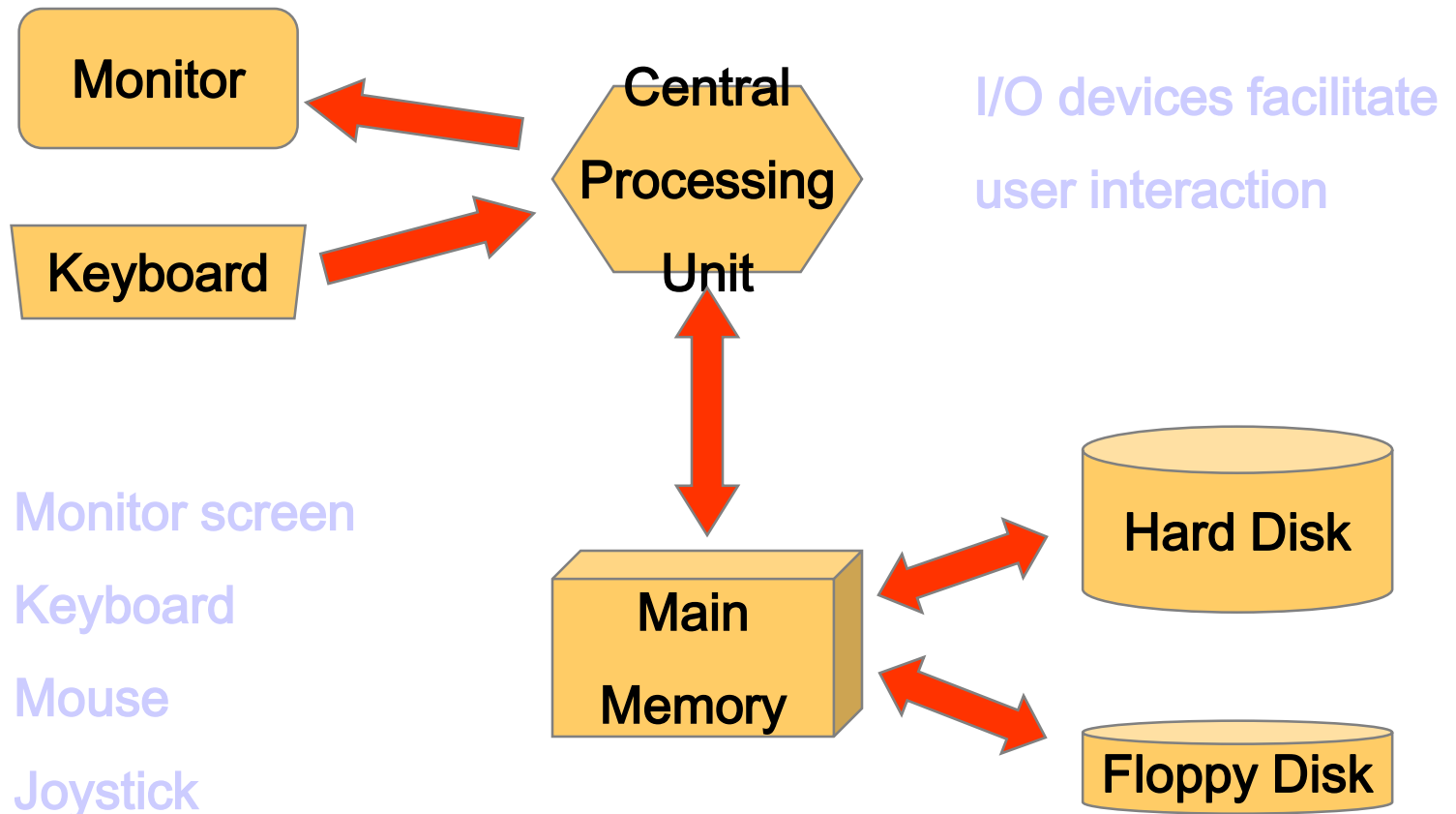
Tapes



Information is moved between main memory and secondary memory as needed



Input / Output Devices



Monitor screen

Keyboard

Mouse

Joystick

Bar code scanner

Touch screen

Analog vs. Digital

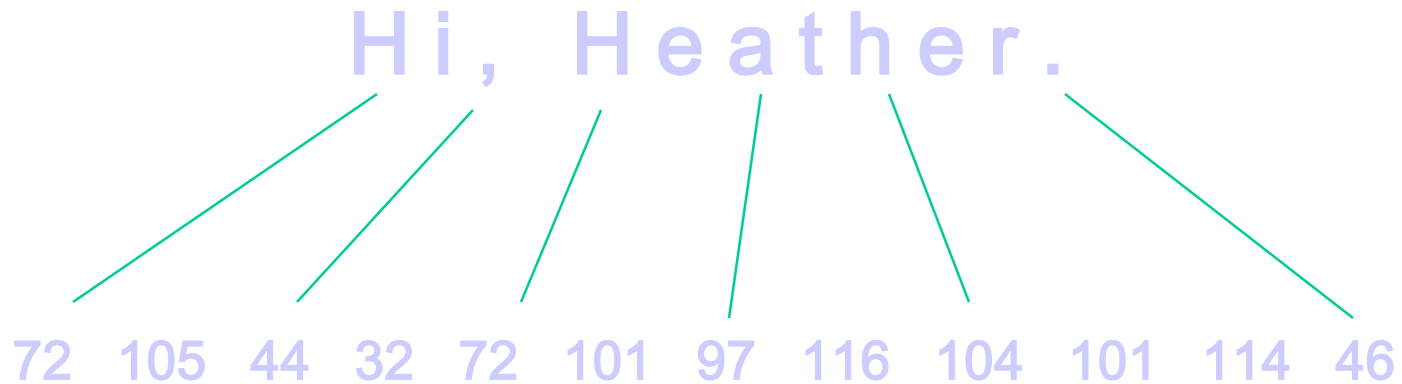
- There are two basic ways to store and manage data:
- *Analog*
 - continuous, in direct proportion to the data represented
 - music on a record album - a needle rides on ridges in the grooves that are directly proportional to the voltages sent to the speaker
- *Digital*
 - the information is broken down into pieces, and each piece is represented separately
 - music on a compact disc - the disc stores numbers representing specific voltage levels sampled at specific times

Digital Information

- Computers store all information digitally:
 - numbers
 - text
 - graphics and images
 - video
 - audio
 - program instructions
- In some way, all information is *digitized* - broken down into pieces and represented as numbers

Representing Text Digitally

- For example, every character is stored as a number, including spaces, digits, and punctuation
- Corresponding upper and lower case letters are separate characters



Binary Numbers

- Once information is digitized, it is represented and stored in memory using the *binary number system*
- A single binary digit (0 or 1) is called a *bit*
- Devices that store and move information are cheaper and more reliable if they have to represent only two states
- A single bit can represent two possible states, like a light bulb that is either on (1) or off (0)
- Permutations of bits are used to store values

Bit Permutations

1 bit

0

1

2 bits

00

01

10

11

3 bits

000

001

010

011

100

101

110

111

4 bits

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

Each additional bit doubles the number of possible permutations

Bit Permutations

- Each permutation can represent a particular item
- There are 2^N permutations of N bits
- Therefore, N bits are needed to represent 2^N unique items

How many
items can be
represented by

1 bit ?

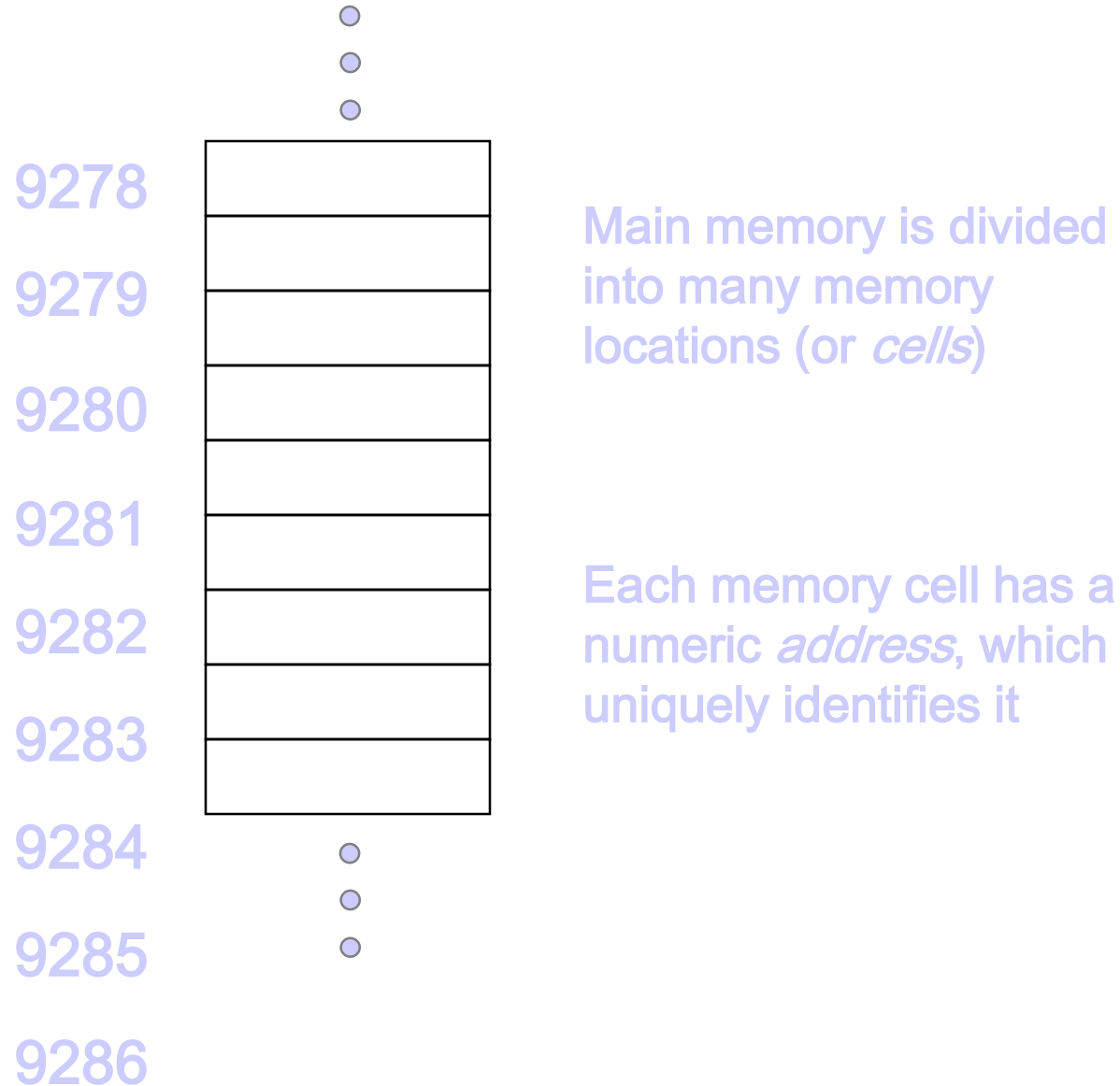
2 bits ?

3 bits ?

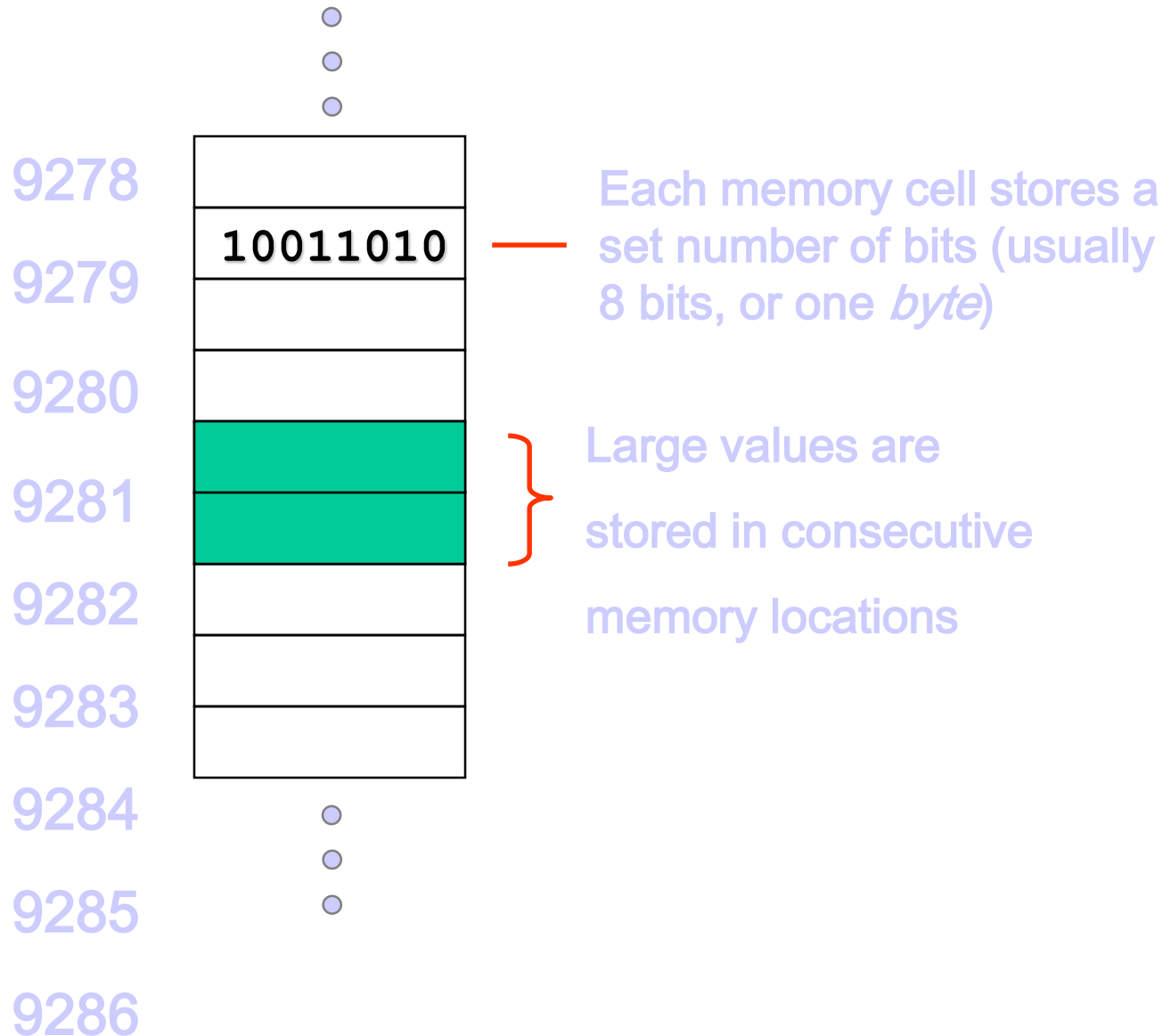
4 bits ?

5 bits ?

Memory



Storing Information



Storage Capacity

- Every memory device has a *storage capacity*, indicating the number of bytes it can hold
- Capacities are expressed in various units:

<u>Unit</u>	<u>Symbol</u>	<u>Number of Bytes</u>
kilobyte	KB	$2^{10} = 1024$
megabyte	MB	2^{20} (over 1 million)
gigabyte	GB	2^{30} (over 1 billion)
terabyte	TB	2^{40} (over 1 trillion)

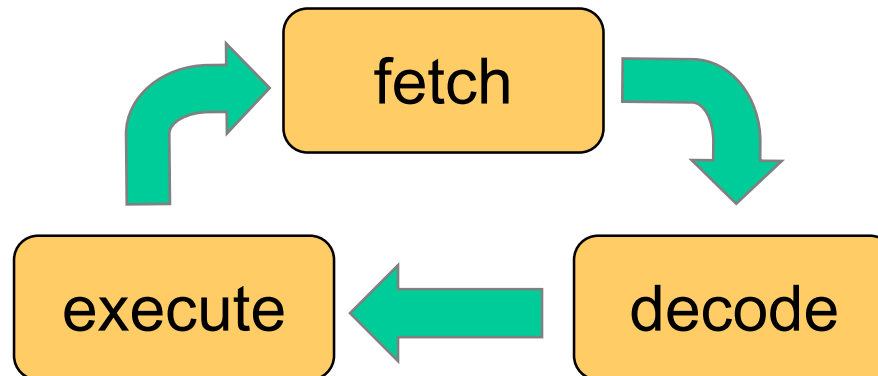
Memory

- Main memory is *volatile* - stored information is lost if the electric power is removed
- Secondary memory devices are *nonvolatile*
- Main memory and disks are *direct access* devices - information can be reached directly
- The terms *direct access* and *random access* often are used interchangeably
- A magnetic tape is a *sequential access* device since its data is arranged in a linear order - you must get by the intervening data in order to access other information

The Central Processing Unit

- A CPU is on a chip called a *microprocessor*
- It continuously follows the *fetch-decode-execute cycle*:

Retrieve an instruction from main memory

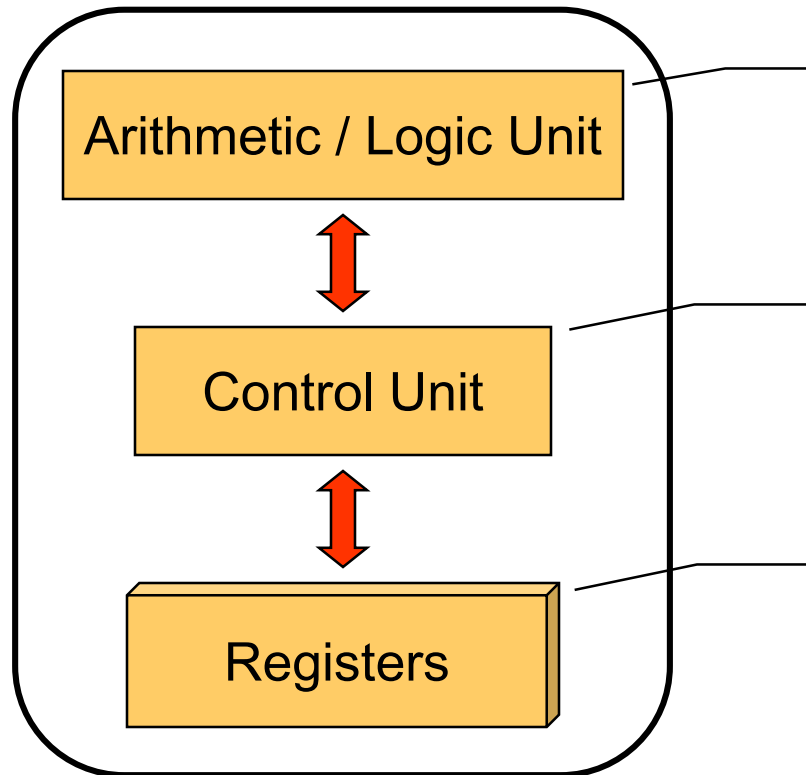


Carry out the
instruction

Determine what the
instruction is

The Central Processing Unit

- The CPU contains:



The Central Processing Unit

- The speed of a CPU is controlled by the *system clock*
- The system clock generates an electronic pulse at regular intervals
- The pulses coordinate the activities of the CPU
- The speed is usually measured in *gigahertz* (GHz)

Monitor

- The size of a monitor (17") is measured diagonally, like a television screen
- Most monitors these days have *multimedia* capabilities: text, graphics, video, etc.
- A monitor has a certain maximum *resolution* , indicating the number of picture elements, called *pixels*, that it can display (such as 1280 by 1024)
- High resolution (more pixels) produces sharper pictures

Language Levels

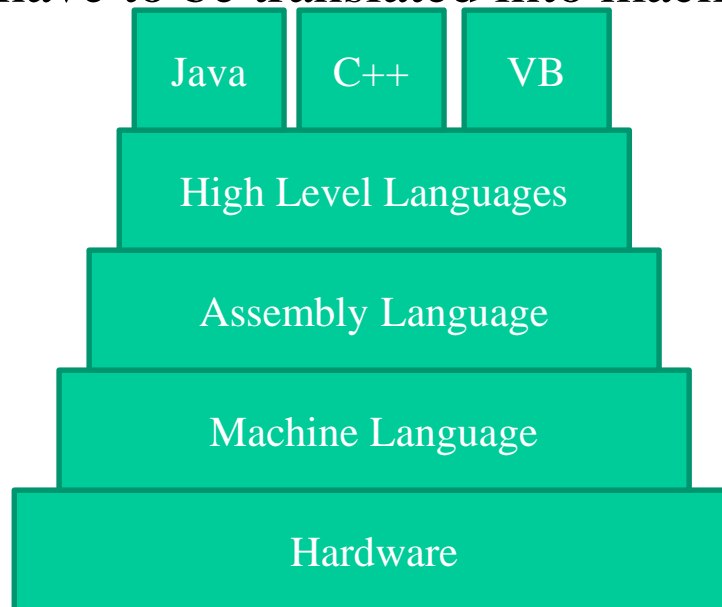
- There are four programming language levels:
 - machine language
 - assembly language
 - high-level language
 - fourth-generation language
- Each type of CPU has its own specific *machine language*
- The other levels were created to make it easier for a human being to read and write programs

Programming Languages

- Each type of CPU executes only a particular *machine language*
- A program must be translated into machine language before it can be executed
- A *compiler* is a software tool which translates *source code* into a specific target language
- Often, that target language is the machine language for a particular CPU type
- The Java approach is somewhat different

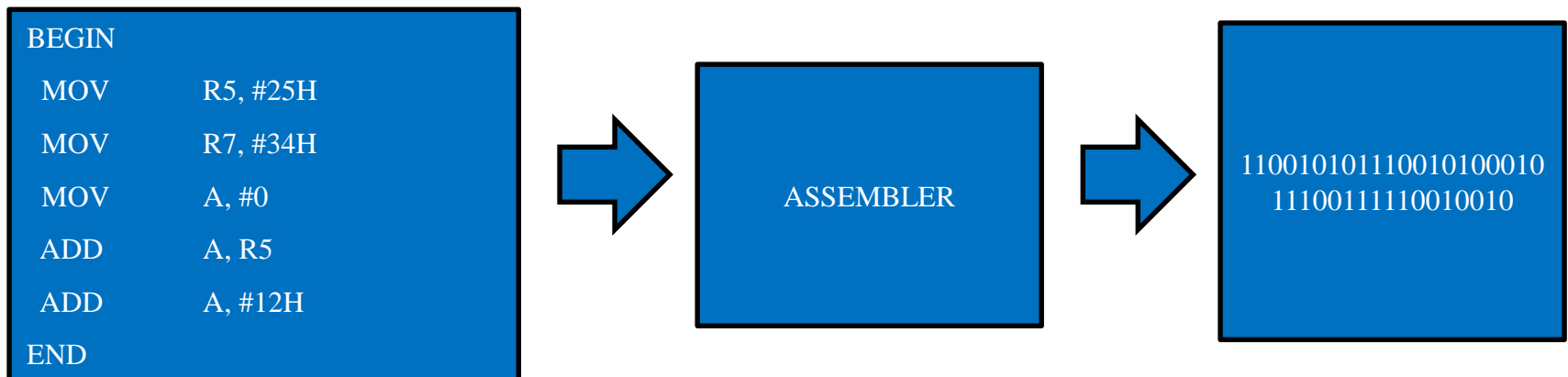
Machine Language

- Computer can execute an instruction only if its in machine language.
- Can the computer understand instructions in other languages?
 - Yes, they have to be translated into machine language.



Assembly Language

- It is a low level programming language.
 - It is very intimate with the machine's architecture
 - I.e. specific to the computer's architecture, as opposed to high-level languages which are generally portable.
- An Assembler
 - Is used to convert the assembly language into machine code



High Level Language(s)

- Almost all programs are written in high level programming languages
 - Uses natural language
 - Automates certain aspects like memory management
 - Make programming simpler and easier to understand than low-level languages
 - Examples: Java, C++, Visual Basic, Pascal, ...
- How are programs built with high level programming languages executed?
 - They have to be translated into machine language to be executed by the CPU.

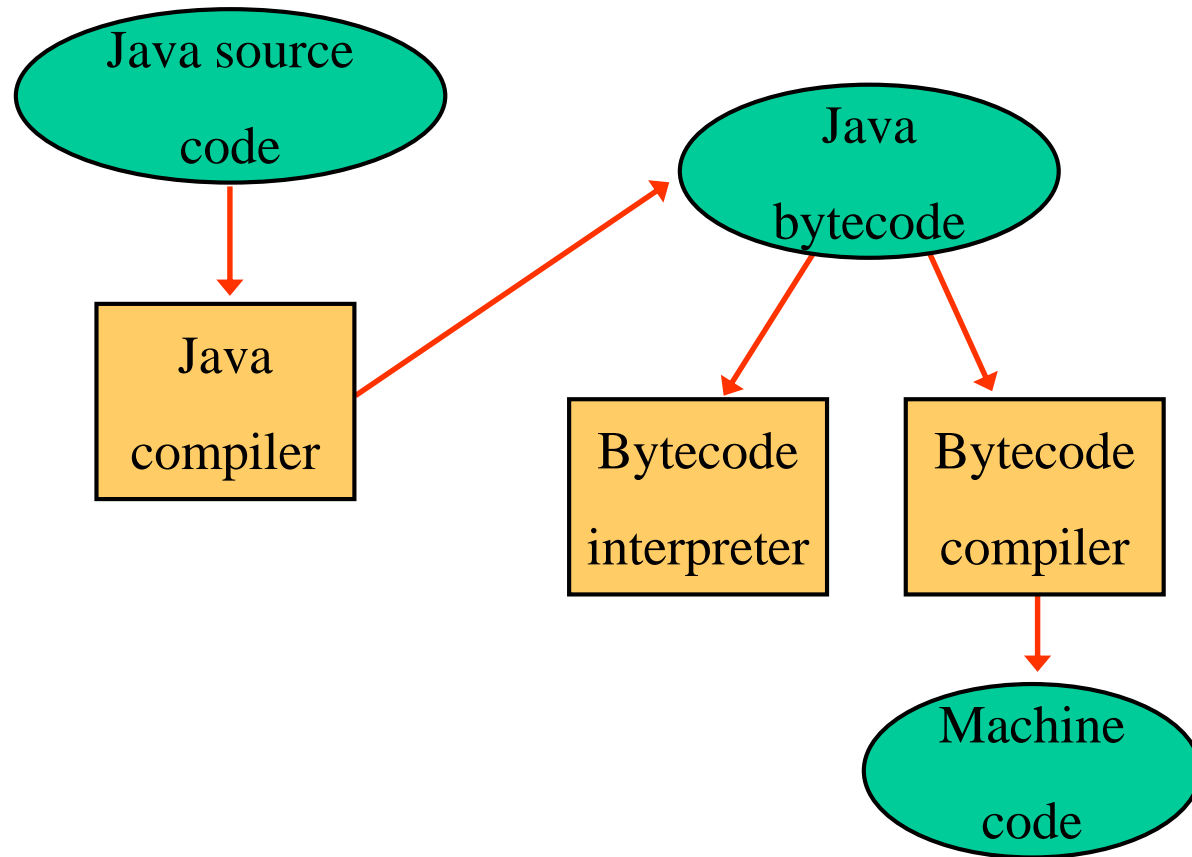
From High level to Low level

- Translation is done by a compiler or interpreter
- Compiler
 - It translates a high-level-language program into an executable machine-language program.
 - Once translated, the machine-language program can be run any number of times.
- Interpreter
 - Translates instruction-by-instruction, as necessary
 - Each time the program has to be executed, the interpreter reads each of its instructions and executes the corresponding machine language equivalent.

Java Translation

- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an *interpreter*, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be *architecture-neutral*

Java Translation



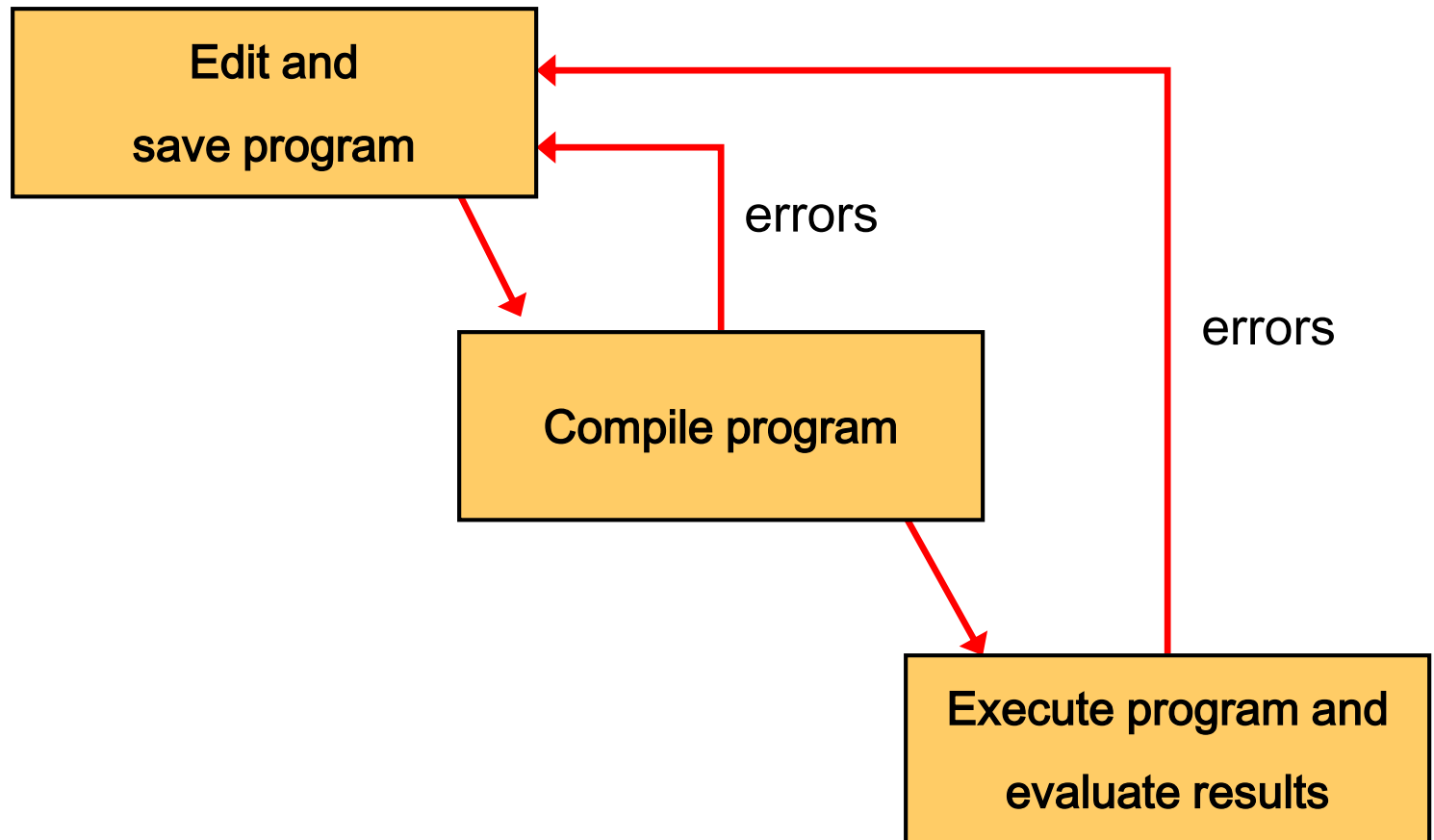
Syntax and Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

Errors

- A program can have three types of errors
- The compiler will find syntax errors and other basic problems (*compile-time errors*)
 - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

Basic Program Development

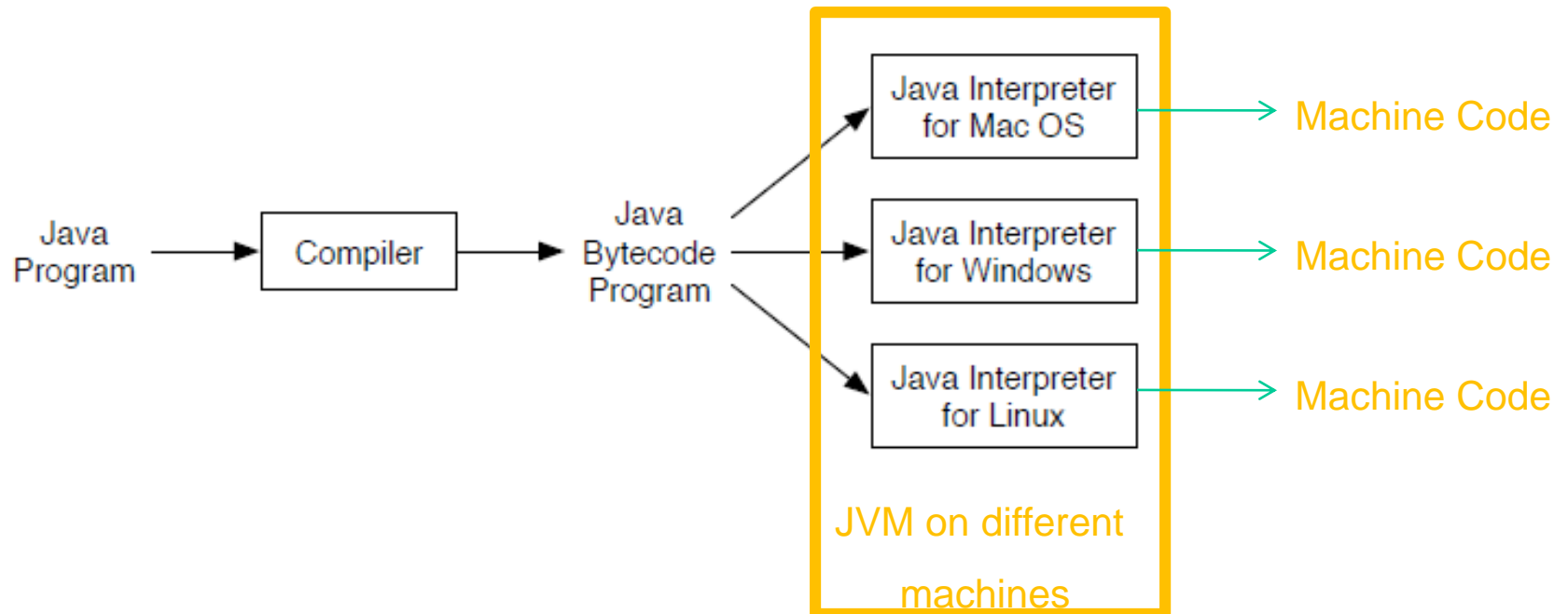


The Java Model

- Java use a combination of compilation and interpretation.
- Java programs are compiled into machine language.
 - But it is a machine language for a computer that doesn't really exist.
 - This so-called “virtual” computer is known as the Java Virtual Machine or JVM.
- The machine language for the Java Virtual Machine is called Java bytecode .
- Advantage of bytecode: its is portable across different types of hardware/machines.

Compile Once, run Anywhere

- Java provides platform independence



Brief History of Java

- In the early 1990's, putting intelligence into home appliances was thought to be the next "hot" technology.
- Examples of intelligent home appliances:
 - Coffee pots and lights that can be controlled by a computer's programs.
 - Televisions that can be controlled by an interactive television device's programs.
- Anticipating a strong market for such things, Sun Microsystems in 1991 funded a research project (code named Green) whose goal was to develop software for intelligent home appliances.
- An intelligent home appliance's intelligence comes from its embedded processor chips and the software that runs on the processor chips.
- Appliance processor chips change often because engineers continually find ways to make them smaller, less expensive, and more powerful.
- To handle the frequent turnover of new chips, appliance software must be extremely portable.

Brief History of Java

- Originally, Sun planned to use C++ for its home appliance software, but they soon realized that C++ was less than ideal because it wasn't portable enough and it relied too heavily on hard-to-maintain things called pointers.
- Thus, rather than write C++ software and fight C++'s inherent deficiencies, Sun decided to develop a whole new programming language to handle its home appliance software needs.
- Their new language was originally named Oak (for the tree that was outside project leader James Gosling's window), but it was soon changed to Java.
- When the home appliance software work dried up, Java almost died before being released.
- Fortunately for Java, the World Wide Web exploded in popularity and Sun realized it could capitalize on that.

Brief History of Java

- Web pages have to be very portable because they can be downloaded onto any type of computer.
- What's the standard language used for Web pages?
- Java programs are very portable and they're better than HTML in terms of providing user interaction capabilities.
- Java programs that are embedded in Web pages are called *applets*.
- Although applets still play a significant role in Java's current success, some of the other types of Java programs have surpassed applets in terms of popularity.
- In this course, we cover Standard Edition (SE) Java *applications*. They are Java programs that run on a standard computer – a desktop or a laptop, without the need of the Internet.

Solving Problems with Computer Programs

- A program is a list of instructions executed by the CPU
- List of instructions to do what ?
 - Basically to solve a problem
- We solve many such trivial problems every day without even thinking about them
 - making breakfast/tea, travelling to the university/workplace
 - solution to the problems above requires little intellectual effort and is relatively unimportant.
- Using a computer to solve problems
 - Implies the steps leading to the solution, must be transmitted to the computer.
 - An algorithm is used for describing the problem before it is converted to instructions.

Problem Solving

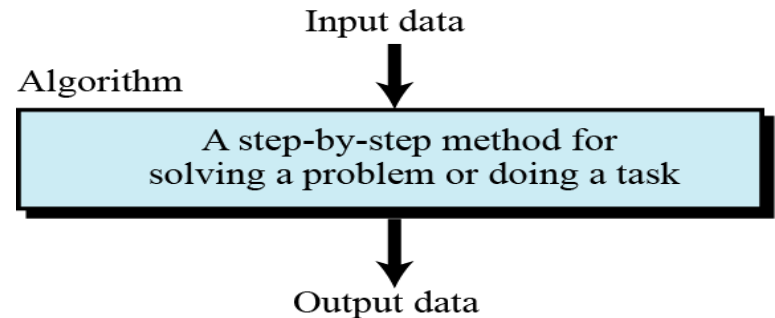
- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
 - Understand the problem
 - Design a solution
 - Consider alternatives and refine the solution
 - Implement the solution
 - Test the solution
- These activities are not purely linear – they overlap and interact

Problem Solving

- The key to designing a solution is breaking it down into manageable pieces
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- An *object-oriented approach* lends itself to this kind of solution decomposition
- We will dissect our solutions into pieces called objects and classes

The Algorithm

- a step-by-step method for solving a problem or doing a task.



- The algorithm can be described on many levels
 - It is just the procedure of steps to take and get the result.
 - It needs to be converted into a computer programming language to be understood by the computer.
 - Irrespective of the language used, the algorithm (series of steps to reach a solution) will be the same.

Algorithm (other definitions)

- A step-by-step problem-solving procedure
- An algorithm is a sequence of unambiguous instructions for solving a problem.
- The number of steps of an algorithm will be countable and finite.
- It is a sequence of instructions (or set of instructions) to make a program more readable; a process used to answer a question.

Algorithm Analogy: An Example

- **Ingredients/Requirements:**

- 1 cup (150g) self-raising flour
- 1/3 cup (50g) cocoa, sifted
- 1 cup (220g) caster sugar
- 1/3 cup (80g) butter, softened
- 1/2 cup (125ml) milk
- 2 eggs, lightly beaten

- **Method/Procedure:**

- Preheat oven to 180°C (160°C fan-forced).
- Grease and flour or grease and line a 24cm cake tin and set aside.
- Place all ingredients into a bowl and using a mixer, mix on high for 4 minutes.
- Pour into cake tin and bake for 35-40 minutes or until the cake springs back when lightly touched in the centre.

Procedure (Method/Function)

- A procedure
 - is a finite sequence of well-defined instructions, each of which can be mechanically carried out in a finite amount of time.
- In the case of a computer
 - the problem solution is usually in the form of a program that encompasses the algorithm and explains to the computer a clearly defined procedure for achieving the solution.
 - The procedure must consist of smaller steps each of which the computers understand.
 - There should not be any ambiguities in the translation of the procedure into the necessary action to be taken.
- A program is just a specific realisation of an algorithm, which may be executed on a physical device.

Understand the problem.

1. Define the problem
2. Analyze the problem
3. Develop an algorithm/method of solution
4. Write a computer program corresponding to the algorithm
5. Test and debug the program
6. Document the program (how it works and how to use it)

- There are two commonly used tools to help to document program logic (the algorithm). These are :

- ❖ Flowcharts

- ❖ Pseudo code.



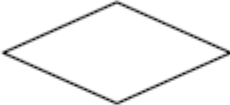



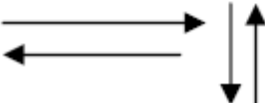
Algorithm Representation

- How to best describe a computer related solution to a problem using an algorithm?
 - We can use flowchart or pseudocode
- Flowchart
 - shows the steps involved in a algorithm, using diagram made up of boxes, diamonds and other shapes, connected by arrows, and each shape represents a step in the process
- Pseudocode (its not real computer code, its pseudo code)
 - It is a generic way of describing an algorithm without use of any specific programming language syntax. It models and resembles real programming code, and is written at roughly the same level of detail.

Flowchart

A **flowchart** is a type of *diagram* that represents an *algorithm* or *process*, showing the steps as boxes of various kinds and their order by connecting these with arrows. This diagrammatic *representation* can give a step-by-step solution to a given *problem*.

Flowcharts

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program.
	Flow Lines	Shows direction of flow.

Flowchart Rules

1. All boxes of the flowchart are connected with Arrows.
2. Flowchart symbols have an entry point on the top of the symbol with no other entry points. The exit point for all flowchart symbols is on the bottom except for the Decision symbol.
3. The Decision symbol has two exit points; these can be on the sides or the bottom and one side.
4. Generally a flowchart will flow from top to bottom. However, an upward flow can be shown as long as it does not exceed 3 symbols.

Flowchart Rules

5. Connectors are used to connect breaks in the flowchart. Examples are:

From one page to another page.

From the bottom of the page to the top of the same page.

An upward flow of more than 3 symbols

6. Subroutines and Interrupt programs have their own and independent flowcharts.

7. All flow charts start with a Terminal or Predefined Process (for interrupt programs or subroutines) symbol.

8. All flowcharts end with a terminal or a contentious loop.

Designing Algorithms & Flowcharts

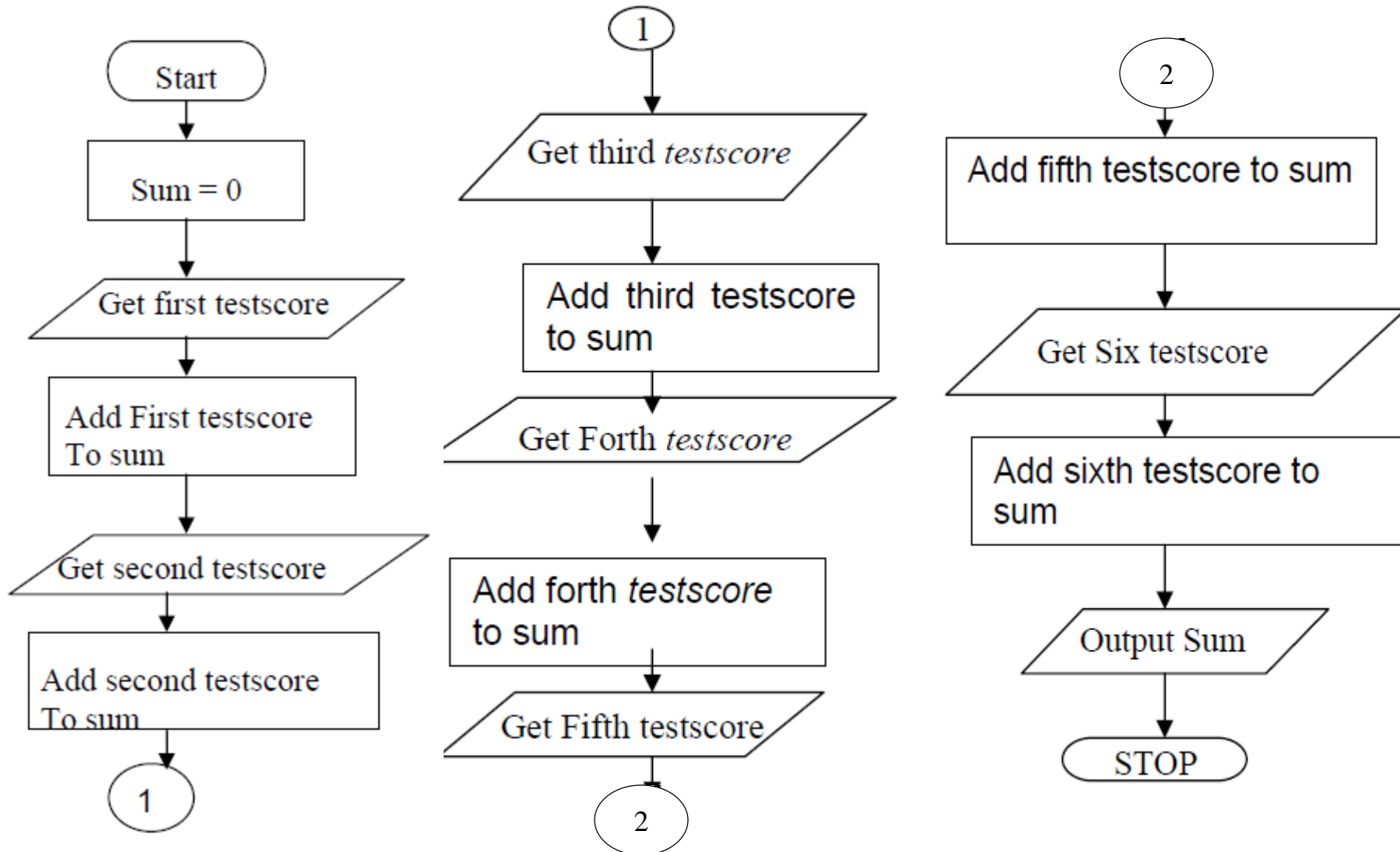
- **Example 1:** Design an algorithm and the corresponding flowchart for adding the test scores as given below:

26, 49, 98, 87, 62, 75

The algorithm:

1. Start
 2. Sum = 0
 3. Get the first testscore
 4. Add first testscore to sum
 5. Get the second testscore
 6. Add to sum
 7. Get the third testscore
 8. Add to sum
 9. Get the Forth testscore
 10. Add to sum
 11. Get the fifth testscore
 12. Add to sum
 13. Get the sixth testscore
 14. Add to sum
 15. Output the sum
 16. Stop
- *avoid writing in two columns

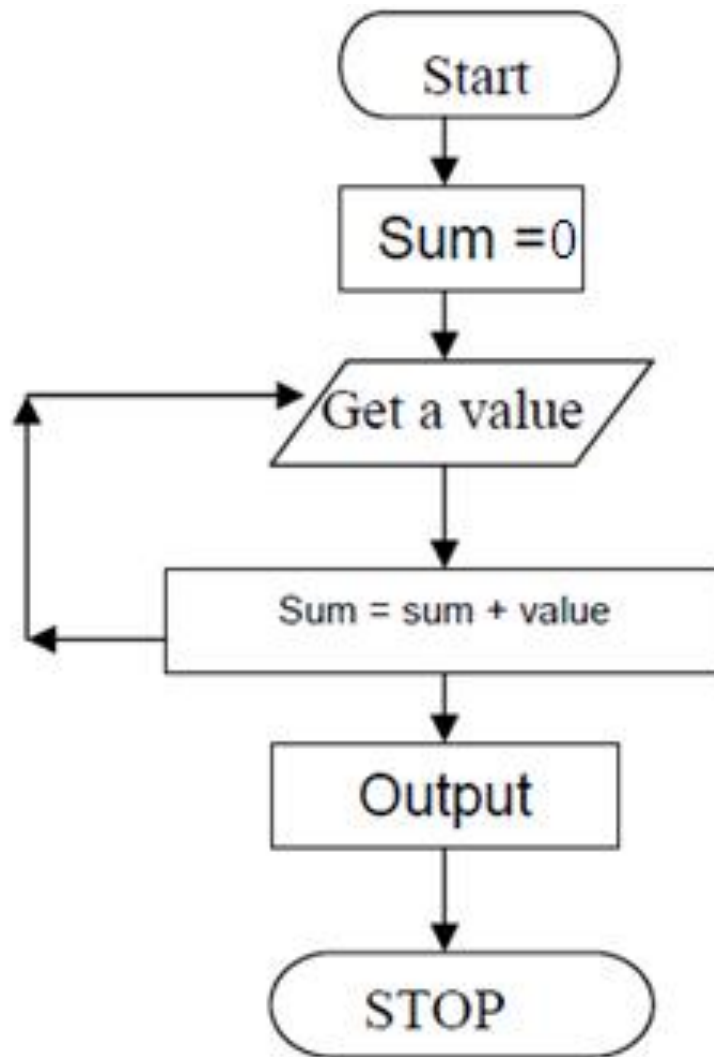
Designing Algorithms & Flowcharts



Refine Example 1 (Algorithm)

- The problem with the algorithm is that, some of the steps appear more than once, i.e. step 5 get second number, step 7, get third number, etc.
- It can be shorten as follows:
 1. Start
 2. $\text{Sum} = 0$
 3. Get a value
 4. $\text{sum} = \text{sum} + \text{value}$
 5. Go to step 3 to get next Value
 6. Output the sum
 7. Stop

Refine Example 1 (Flowchart)



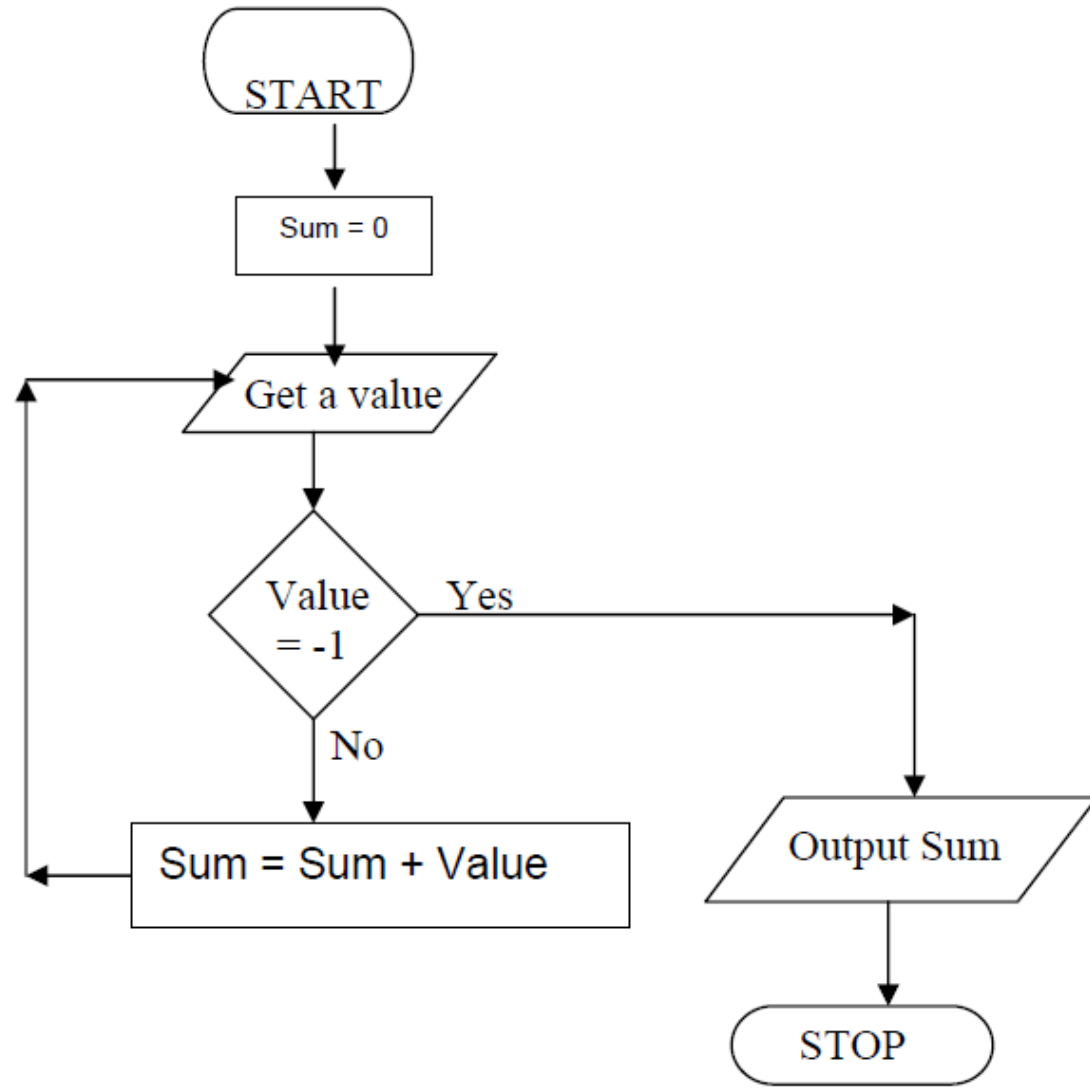
Have you noticed any problem with this new algorithm and its flowchart ?

How to make the solution finite?

- The steps will be repeated endlessly, resulting in an endless algorithm or flowchart.
- A solution to this problem is to add a unique last value to the list of numbers given:
26, 49, 98, 87, 62, 75, -1
- Write down the update algorithm and draw the flowchart to represent the updated solution.

The (finite) Solution

1. Start
2. $\text{Sum} = 0$
3. Get a value
4. If the value is equal to -1 , go to step 7
5. Add to sum ($\text{sum} = \text{sum} + \text{value}$)
6. Go to step 3 to get next Value
7. Output the sum
8. Stop



Pseudocode

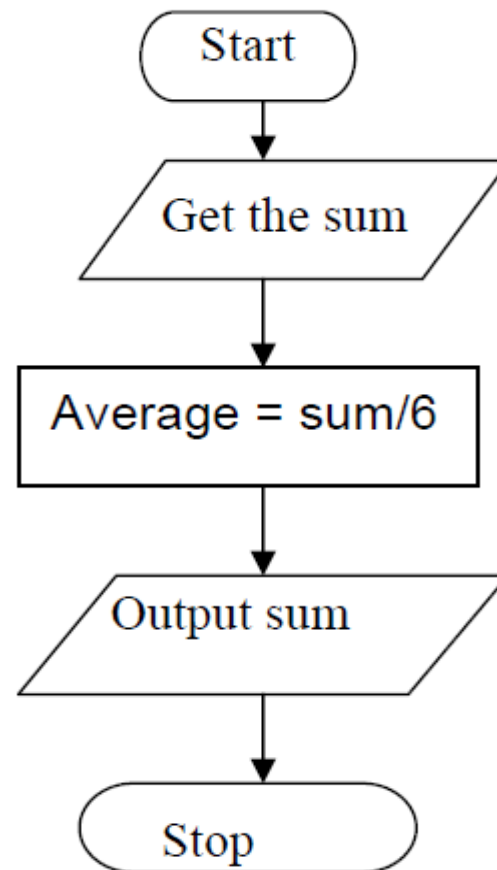
- Pseudocode exists in various forms, although most of them uses popular programming language syntax.
- In algorithm design, the steps of the algorithm are written in free English text
 - Short sentences as far as possible
 - but at times they can be long in order to describe the particular operation.
 - The steps of an algorithm are said to be written in pseudocode.
- It actually makes the transition from design to coding extremely easy.

Pseudocode Example

- Example 2 : design an algorithm for finding the average of six numbers, and the sum of the numbers is given.

1. Start
2. Get the sum
3. $\text{Average} = \text{sum} / 6$
4. Output the average
5. Stop

Pseudocode



flowchart

Pseudocode Example

- **Example 3:** Pseudocode required to input three numbers from the keyboard and output their sum
 1. Use variables: sum, number1, number2, number3 of type integer
 2. Input number1, number2, number3
 3. $\text{Sum} = \text{number1} + \text{number2} + \text{number3}$
 4. Print sum
 5. End program
- Its subtle, but its closer to the way we write a computer program (notice words like variable, type, integer, print)

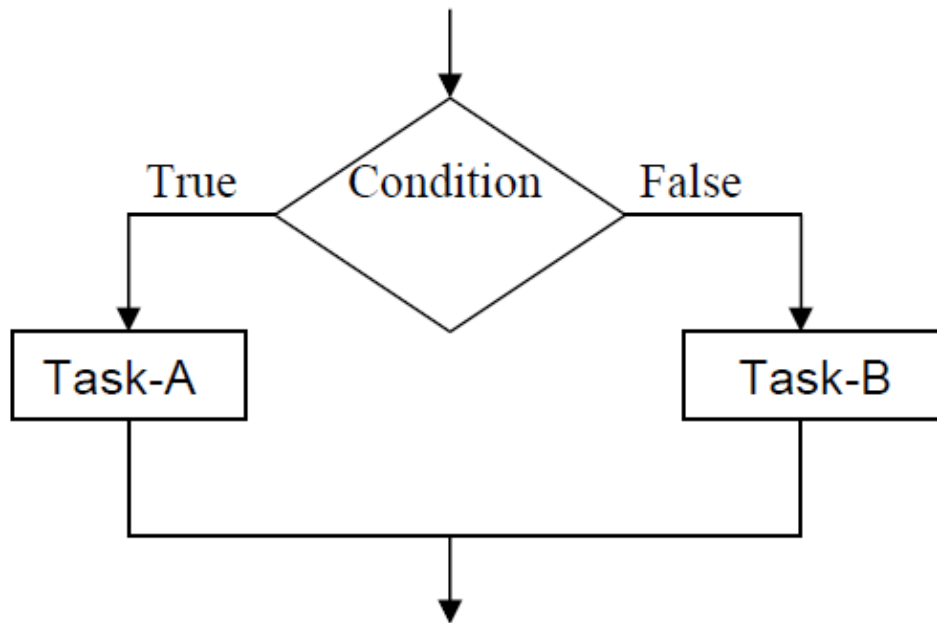
Pseudocode Example

- **Example 4:** Write pseudo-code to describe an algorithm which will accept two numbers from the keyboard and calculate the sum and product displaying the answer on the monitor screen.
 1. Use variables sum, product, number1, number2 of type real
 2. display “Input two numbers”
 3. input number1, number2
 4. $\text{sum} = \text{number1} + \text{number2}$
 5. print “The sum is “, sum
 6. $\text{product} = \text{number1} * \text{number2}$
 7. print “The Product is “, product
 8. end program
- We notice even more resemblance to a computer program.

Sequence, Condition, Iteration

- Lets take some time and think about program flow
 - Is it always sequential ?
 - All parts of the program have to be executed ?
 - Can computers take decisions?
 - If, yes, how should programs be written to support decision making?
 - What happens when there is a procedure which needs to be repeated several times?
- Actually the following flow in the structure of a program is supported:
 - Sequential (one after the other)
 - Conditional/decision making (depending on a situation)
 - Iteration/Repetition (process same instructions several times)

Decision in Pseudocode



- Pseudocode

If *condition* is
true
Then do task A
Else
Do Task-B

Making Choices

- The logical operators which are understood by the CPU and which are used in pseudocodes:
 - = is equal to
 - > is greater than
 - < is less than
 - >= is greater than or equal
 - <= is less than or equal
 - <> is not equal to

Decision in Pseudocode

Example 1:

Use variables: money of type real

Input money

If money > 200

 display “can go to KFC”

Else

 display “go to university
 canteen”

Example 2:

Use variables: mark of type integer

Input mark

If mark >= 80

 display “distinction”

Else If mark >= 60 AND mark < 80

 display “merit”

Else If mark >= 40 AND mark < 60

 display “pass”

Else If mark < 40

 display “fail”

Compound Logical Operators

- At several occasions, conditions need to be linked.
 - If I had time **and** money, I would go on a holiday
 - I am happy to go to the beach **or** the cinema

- Examples

If $a > b$ AND $a > c$

display “a is the largest”

Else

display “a is not the largest”

If movie = “transformers” OR

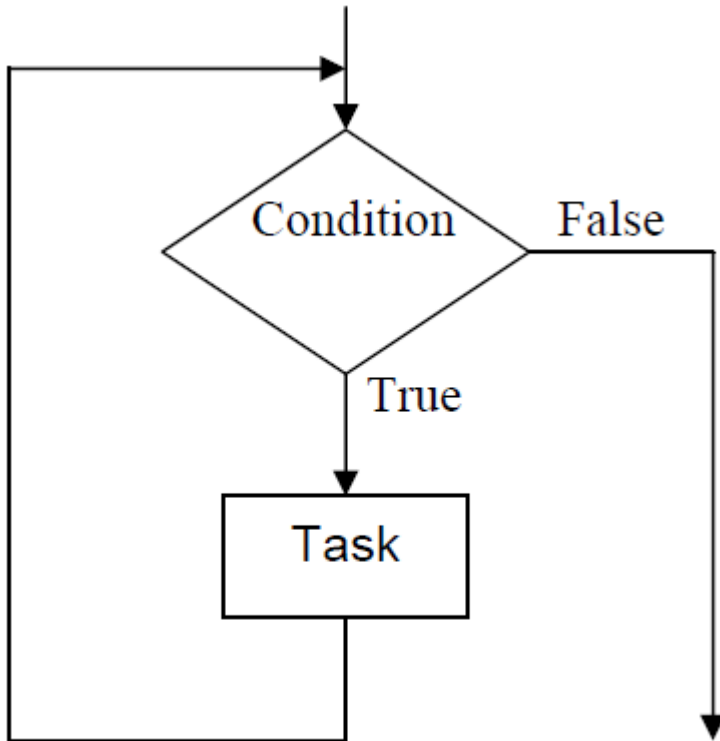
movie = “Iron man”

display “go to cinema”

Else

display “go to the beach”

Repetition / Iteration in Pseudocode



- In pseudocode, repetition is supported basically in three modes:
 - Repeat until loop
 - While loop
 - For loop

Repeat Until Loop

REPEAT

A statement or block of statements

UNTIL a true condition

REPEAT

DISPLAY “Enter a number between 1 and 100”

Input number

UNTIL (number < 1 OR number > 100)

Do While

WHILE (a condition is true)

A statement or block of statements

ENDWHILE

WHILE (letter <> 'q')

 READ letter

 DISPLAY “The character you typed is”, letter

ENDWHILE

For Loop

FOR (starting state, stopping condition, increment)

Statements

END FOR

FOR (n = 1, n <= 4, n + 1)

DISPLAY “loop”, n

END FOR

Example

- Design an algorithm and the corresponding flowchart for finding the sum of n numbers.
- *Pseudocode Program*

Start

sum = 0

Display “Input value n”

Input n

For(i = 1, i <= n, i+1)

Input a value

sum = sum + value

ENDFOR

Output sum

Stop

Exercises

1. Draw the flowchart for the last (example) algorithm that calculates the sum of n numbers, and write it using a Repeat Until loop and then with a While loop.
2. Design an algorithm and the corresponding flowchart for finding the sum of the numbers 2, 4, 6, 8, ..., n
3. Using flowcharts, write an algorithm to read 100 numbers and then display the sum.
4. Write an algorithm to read two numbers then display the largest.
5. Write an algorithm to read two numbers then display the smallest.
6. Write an algorithm to read three numbers then display the largest.
7. Write an algorithm to read 100 numbers then display the largest.

Java

- A *programming language* specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
- The Java programming language was created by Sun Microsystems, Inc.
- It was introduced in 1995 and its popularity has grown quickly since

Java Program Structure

- In the Java programming language:
 - A program is made up of one or more *classes*
 - A class contains one or more *methods*
 - A method contains program *statements*
- These terms will be explored in detail throughout the course
- A Java application always contains a method called `main`

Java Program Structure

```
// comments about the class
```

```
public class MyProgram
```

```
{
```

class header

class body

Comments can be placed almost anywhere

```
}
```

Java Program Structure

```
// comments about the class
```

```
public class MyProgram
```

```
{
```

```
    // comments about the method
```

```
    public static void main (String[] args)
```

```
    {
```



method body



method header

```
    }
```

```
}
```

```
//*****//
Lincoln.java          Author: Lewis/Loftus

//  Demonstrates the basic structure of
//    Java application.

//*****

public class Lincoln{

    //-----

    //  Prints a presidential quote.

    //-----

    public static void main (String[] args){

        System.out.println ("A quote by Abraham Lincoln:");

        System.out.println ("Whatever you are, be a good
one.");

    }

}
```

Comments

- Comments in a program are called *inline documentation*. A note written in the source code by the programmer to make the code easier to understand.
 - Comments are not executed when your program runs.
 - Most Java editors turn your comments a special color to make it easier to identify them.
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line
```

```
/* this comment runs to the terminating  
   symbol, even across line breaks */
```

```
/** this is a javadoc comment */
```

Identifiers

- *Identifiers* are the words a programmer uses in a program. A name that we give to a piece of data or part of a program.
 - Identifiers are useful because they allow us to refer to that data or code later in the program.
 - Identifiers give names to:
 - classes
 - methods
 - variables (named pieces of data; seen later)
- An identifier can be made up of letters, digits, the underscore character (`_`), and the dollar sign
- Identifiers cannot begin with a digit. first character must a letter or `_` or `$`
- Java is *case sensitive* - `Total`, `total`, and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers, such as
 - *title case* for class names - `Lincoln`
 - *upper case* for constants - `MAXIMUM`

Identifiers

- Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that they chose (such as `println`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

Reserved Words/Keywords

- The Java **keyword**: An identifier that you cannot use, because it already has a reserved meaning in the Java language.

• abstract	else	int	strictfp
boolean	enum	interface	super
break	extends	long	switch
byte	false	native	synchronized
case	final	new	this
catch	finally	null	throw
char	float	package	throws
class	for	private	transient
const	goto	protected	true
continue	if	public	try
default	implements	return	void
do	import	short	volatile
double	instanceof	static	while

White Space

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted many ways
- Programs should be formatted to enhance readability, using consistent indentation