# Introduction to Technical Programming

# Module 1: Computer hardware and software

1.1 Different types of hardware of a common system

**Content:**

• System Unit

• Motherboard

• CPU

• Memory

• Input and Output hardware

Learning Outcomes:

Students should be able to:

1.1.1 Identify the components of the system unit

1.1.2 Explain the term CPU and its purpose

1.1.3 Describe the term CPU and explain the impact of using various types of CPU’s

1.1.4 Describe different types of CPUs regarding use, and power

1.1.5 Define the term computer memory

1.1.6 Discuss the **primary purpose of memory**

**1.1.7 Differentiate between different types of memory and their purpose (Range: RAM, ROM, CMOS, Cache memory, Flash Memory)**

**1.1.8 Explain how data is stored** on memory

1.1.9 Define the purpose of the motherboard and its components

1.1.10 Describe different types of input hardware regarding use and classification e.g., direct, and indirect entry

1.1.11 Discuss how data is transferred between memory i.e., primary, and secondary and the CPU

1.1.12 Describe the Flow/transfer of data between components. (Range: USB – PnP, U3, Point-to- point connections)

* + 1. Describe the factors to consider when choosing an input device. (Range: Ergonomic considerations, Wireless vs cables)
    2. Describe different types of output hardware
    3. Describe the purpose and use of devices such as docking stations for mobile and laptop computers.

1.2 **Purpose of software**

Content:

* Software as a component of a computerised system
* Categories of software

Learning Outcomes:

*Students should be able to:*

* + 1. Describe the term software
    2. Describe the purpose and function of software
    3. Describe the basic concepts of software. (Range: Software as programs. Identify software components. Concept of a graphical user interface (GUI).)
    4. Contrast: System software vs application software
    5. Differentiate between: Shareware, Freeware, Open-Source Software and Proprietary software, Firmware
    6. Discuss the process of how software is obtained and installed.
    7. Differentiate between online software and installed software.
    8. Discuss the following terms in relation to software. (Range: Compatibility issues, Versions, patches, and service packs, Updating software)

1.3 **The Linux shell**

Content:

• Working with the Linux Bash terminal

Learning Outcomes:

Students should be able to:

* + 1. Launch a new Linux terminal on the Raspberry Pi
    2. Use the man command to get help
    3. Expand a Linux file path and explain each element
    4. List the contents of the current folder using the ls command
    5. Change directly location using the cd command
    6. Create a new folder using the mkdir command
    7. Remove a folder using the rmdir command
    8. Remove a file using the rm command
    9. Rename a file using the mv command
    10. Copy a file using the cp command
    11. Clear the command prompts screen using the cls command
    12. Run an executable file from the command line

# Module 2 Problem solving in computer programming

2.1 **Problem solving process and concepts**

Content:

* Problem solving
* Application of problem-solving constructs
* Developing solutions

*Learning Outcomes:*

*Students should be able to:*

2.1Define the term problem solving

2.1.2 Define the term computational thinking

2.1.3 Describe the phases of the PLDC (Program Development Life Cycle)

2.1.4 Describe the purpose of problem solving leading to solutions

2.1.5 Explain and apply various problem-solving steps. Polya, G., 1957) (Range:Understand the problem (task/problem description or scenario/user stories) State in own words Clarity on what needs to be done

What is known or given?

What is missing or needed?

Devise a plan/algorithm (storyboard – visual or textual)

Look for patterns

Look at related problems, known solutions

Examine simpler or special cases

Make a table, create diagram, use guess and check, work backwards, identify sub-goal

Carry out the plan/implement the algorithm (write the code)

Look back/test (see if it works)

Check results against original problem. Does it make sense? Is there another solution?)

2.1.6 Use appropriate tools and techniques to present a solution. Range:

User stories (written by the client and provide the requirements)

Noun-verb analysis of user stories

List of nouns provides identification of objects and state

List of verbs provides identification of behaviour

Acceptance tests (does the program meet the requirements?)

2.2 **Construct an algorithm and present a solution to a given problem**

Content:

* Problem solving
* Algorithm design
* Flowcharts

*Learning Outcomes:*

*Students should be able to:*

2.2.1 Define the term algorithm and its purpose in the problem-solving process. (Range: Basic concepts of an algorithm. What is an algorithm? Develop a clear understanding of the problem presented.)

2.2.2 Implement and understand the basic algorithmic constructs used to create a **flowchart.** Range: Input, Output, Processing and Calculations, Selection Iteration

2.2.3 Create a flowchart to present a particular algorithm and its associated tasks

2.2.4 Interpret a basic flow chart and describe its intended operation / function

# Module 3:Concepts of programming for single board microprocessor or microcontrollers

**3.1 Introduction to IO on single board computing**

Content:

* GPIO
* Physical computing

Learning Outcomes:

Students should be able to:

3.1.1 Exploring the Arduino board

3.1.2 Expand the term GPIO

3.1.3 Compare and contrast some of the major advantages of python compared to other programming languages

3.1.4 Explain and identify where the GPIO pins are located

3.1.5 Differentiate between a compiler and an interpreter

3.1.6 Explain the purpose of the GPIO pins

3.1.7 Discuss the major characteristics of the python programming language as an interpreted one

3.1.8 Define the term physical computing

3.1.9 Differentiate between a shell and an IDE

3.1.10 Read and interpret a Pi GPIO Pin guide

3.1.11 Define the term physical computing

3.1.12 Explain what the terms 3V3, 5V, GND GP2 means on the GPIO board

3.1.13 Discuss how Arduino is used to enable physical computing

**3.2 Visual Programming and solution development**

Content:

• Writing program code using a graphical (visual) programming language

• Debugging

• Compilation

• Testing

• Running an application

*Learning Outcomes:*

*Students should be able to:*

3.2.1 Construct (code/write) using the visual tool, debug and run simple programs incorporating: Declaration of variables of different types, use and assignment of values to variables, incorporating program constructs with sequence, selection and iteration structures. Expose and apply various programming concepts as part of the coded solution such as:

3.2.1.1 retrieving remainders: modulus

3.2.1.2 differentiate between real value division and integer division

3.2.1.3 comparison operators and performing logical comparisons

3.2.1.4 incorporate and write code constructs to perform basic calculations such as area, volume, VAT and simple formulae, typical calculations done in other subjects

3.2.1.5 include conditional constructs [if and ifthen-else] (up to a maximum of two nested levels)

3.2.1.6 Include iteration (looping) structures [fixed counter loop]

3.2.1.7 incorporate a combination of iteration and condition structures as part of the solution (i.e. program code)

3.2.2 Write code which applies programming language tools and constructs to draw various shapes (turtle type commands) on an output screen/window. Reinforce concepts such as:

3.2.2.1 Sequence

3.2.2.2 Selection

3.2.2.3 Iteration

3.2.2.4 Creation of objects and shapes

3.2.3 Design a coding solution to a problem incorporating a combination of different programming constructs which include:

3.2.3.1 Sequence

3.2.3.2 Selection

3.2.3.3 Iteration

3.2.4 Design and develop solutions for specific problems that include computational thinking and applying software engineering principles.

3.2.5 Explore lists/arrays (storing and accessing a list of numbers and strings) and containers. (Range: Manipulating lists/arrays such as adding, deleting, replacing, inserting items.)

# Module 4 Programming tools and utilities

* 1. **Building and running C/C++ applications**

Content:

* The C and C++ compiler
* Compiler artefacts
* Compiler design

*Learning* Outcomes*:*

*Students should be able to:*

* + 1. Define *the* term compiler
    2. Define the term source code
    3. Explain what a decompiler is used for
    4. Define the term interpreter
    5. Explain the difference between a compiler and an interpreter
    6. Explain what a binary is and when it is produced
    7. Explain the difference been C and C++ compiler
    8. List and Explain the basic three stage compiler design (Front Middle Back end)
  1. **C/C++ on the Desktop**

Content:

* C/C++ compiler environment on desktop PC
* IDE on desktop PC

*Learning Outcomes:*

*Students should be able to:*

* + 1. Install and configure C/C++ compiler on desktop PC
    2. Define the term IDE
    3. Explain what an IDE is used for
    4. Install and configure IDE on desktop PC

# Module 5 Introduction to a high-level programming language

5.1 Creating a Hello World application

Content:

* Geany IDE
* Compiling
* Building
* Running

Learning Outcomes:

Students should be able to:

5.1.1 Create a new project using an IDE

5.1.2 Create a new file with extension CPP

5.1.3 Explain why C++ source files use the .cpp extension and C source files uses the .c extension.

5.1.4 Write the C/C++ code to display output “Hello World”.

5.1.5 Compile a C/C++ application using an IDE

5.1.6 Explain what type of file is produced by the C/C++ compilation process

5.1.7 Show or hide the IDE message window

5.1.8 Investigate the IDE message window to determine if compilation succeeded

5.1.9 Investigate the IDE message window to locate line numbers that contain errors

5.1.10 Build a C/C++ application using an IDE

5.1.11 Explain what file type is produced by the C/C++build process

5.1.12 Run a C/C++ application using the IDE

5.2 C++ application structure

Content:

* C/C++ fundamentals
* Comments

5.2.1 List and explain the different parts of a simple C/C++ application

5.2.1.1 Headers

5.2.1.2 Name space

5.2.1.3 Main

5.2.1.4 return

5.2.2 Explain case sensitivity as it applies to C/C++

5.2.3 Explain the term free-format language and how that applies to C/C++

5.2.4 Explain the term block-structured language and how that applies to C/C++

5.2.5 Explain what a comment is used for in the C/C++ language

5.2.6 Explain how the compiler will treat a comment

5.2.7 List and explain two types of comments allowed by the C/C++ compiler

5.2.8 Discuss the importance of adding comments to a C/C++ application

# Module 1: Computer hardware and software

1.1 Different types of hardware of a common system

After you have completed this module, you should be able to :

* Identify the components of the system unit
* Explain the term CPU and its purpose
* Describe the term CPU and explain the impact of using various different types of CPU’s
* Describe different types of CPUs with regard to use, and power
* Define the term computer memory
* Discuss the primary purpose of memory
* Differentiate between different types of memory and their purpose (Range: RAM, ROM, CMOS, Cache memory, Flash Memory)
* Explain how data is stored on memory
* Define the purpose of the motherboard and its components
* Describe different types of input hardware with regard to use and classification e.g. direct and indirect entry
* Discuss how data is transferred between memory i.e. primary and secondary and the CPU
* Describe the Flow/transfer of data between components. (Range: USB – PnP, U3, Point-to-point connections)
* Describe the factors to consider when choosing an input device. (Range: Ergonomic considerations, Wireless vs cables)
* Describe different types of output hardware
* Describe the purpose and use of devices such as docking stations for mobile and laptop computers.
* Describe the term software
* Describe the purpose and function of software
* Describe the basic concepts of software. (Range: Software as programs. Identify software components. Concept of a graphical user interface (GUI).)
* Contrast: System software vs application software
* Differentiate between: Shareware, Freeware, Open Source Software and Proprietary software, Firmware
* Discuss the process of how software is obtained and installed.
* Differentiate between online software and installed software.
* Discuss the following terms in relation to software. (Range: Compatibility issues, Versions, patches and service packs, Updating software)
* Launch a new Linux terminal on the Raspberry Pi
* Use the man command to get help
* Expand a Linux file path and explain each element
* List the contents of the current folder using the ls command
* Change directly location using the cd command
* Create a new folder using the mkdir command
* Remove a folder using the rmdir command
* Remove a file using the rm command
* Rename a file using the mv command
* Copy a file using the cp command
* Clear the command prompts screen using the cls command
* Run an executable file from the command line

# 1.1 Different types of hardware of a common system

### Introduction

Our daily lives have become increasingly reliant on computers for everyday tasks. As a result, computer programming has become an essential skill. Even though not everyone has the skill, those who choose to be programmers must understand how computers work. Knowledge of computer architecture increases the understanding of some programming concepts, not simply knowing them. This module introduces the types of hardware to begin with a basic understanding of how a system functions. In general terms, there are only four basic functions that a computer can perform through the hardware and software and these are:

* **Data processing-** Data can take many forms, and the processing requirements are numerous.
* **Data storage-** Even if the computer is processing data on the fly, it must temporarily store at least those pieces of data that are currently being worked on.
* **Data movement-** The operating environment of a computer is made up of devices that act as data sources or destinations.
* **Control-** A control unit manages the computer's resources and orchestrates the performance of its functional parts in response to commands.

### 1.1.1 Identify the components of the system unit

**VOCABULARY**

A system unit is the part of a computer that houses the primary devices that perform operations and produce results for complex calculations.

In this section, we will explain what a system unit is and its purpose on a computer. The system unit contains components such as the motherboard, the central processing unit (CPU), random access memory (RAM), and other devices. This unit performs most of the tasks that a computer is required to perform. In general, system units are used to differentiate between the computer itself and its peripheral devices, such as the monitor, keyboard, and mouse. In layman's terms, a system unit is also called a chassis or a tower. Figure 1.1 illustrates some components making up a system unit. Some of the components are too small to show diagrammatically and we will try to explain them in detail.

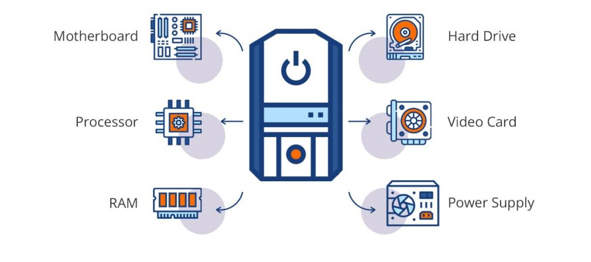


Figure 1. 1: System Unit components

**General functions of components**

Providing proper system functioning is dependent on each component's unique functions. Here are some of the general functions of the main components:

* **Motherboard**. Since connecting all computer nodes to one another is this device's primary function, it is essentially just a series of cables connecting the processor and memory modules.
* **Processor** The computer's "brain," or core, or the element that handles the majority of the work is called the processor. The processor receives data from other parts of the computer and uses it to make computations. The data is returned to the many devices that utilise it after mathematical calculations.
* **RAM** Data storage and quick access are goals of RAM. After the power is cut off, all data on it is lost. Therefore, RAM is referred to being volatile. Volatile memory is a type of memory that maintains its data only while the device is powered. If the power is interrupted for any reason, the data is lost.
* Hard Drive- The hard drive is intended to serve as the computer's permanent storage for files such as music, movies, pictures, and documents.
* **Video Card** converts the image in the computer’s memory into a video signal for the monitor.
* **Power Supply** provides power to the PC. It also performs the functions of stabilization and protection against minor interference in the electrical network.

### 1.1.2 Explain the term CPU and its purpose

**VOCABULARY**

The computer's central processing unit (CPU) is the portion of a computer that retrieves and executes instructions. The CPU is essentially the brain of a computer system.

The acronym CPU stands for Central Processing Unit. The CPU is placed into a specific square-shaped socket found on all motherboards by inserting its metallic connectors or pins found on the underside. Each socket is built with a specific pin layout to support only a specific type of processor.

The CPU is the part of a computer that handles the data and activities of the various physical components of the computer. It transfers instructions between a computer's hardware and software. It is also known as a processor, microprocessor, or central processor. Inputs enter a computer and travel to the CPU where they get processed to produce output.

### 1.1.3 Describe the term CPU and explain the impact of using various types of CPU’s

The CPU executes the instructions and delivers the results to the associated output. Basically, the CPU is the heart of a computer. It takes in the necessary information and processes it, thus allowing the computer to function. A CPU has three main parts: an arithmetic logic unit (ALU), a control unit (CU), and a memory unit. Based on the von Neumann architecture, the CPU is equal to the brain of a computer since it controls everything in the computer system. John von Nuemann is a Hungarian American mathematician, physicist, computer scientist, engineer. He made a great contribution to computer architecture.

**Instruction Processing cycle**

When a program is being carried out, the CPU implements the fetch-decode – execute cycle, which recurs repeatedly until arriving at the STOP instruction. Here are the common five stages of the fetch-decode-execute cycle.

1. **Fetch- Instruction from Memory (Instruction Fetch, IF)**

Each instruction is stored in memory and has its own address. The processor takes this address number from the program counter, which is responsible for tracking which instructions the CPU should execute next. Once the instruction is fetched, the program counter (PC) will have the address of the next instruction to be executed. A program counter is also known as an instruction counter, instruction pointer, instruction address register or sequence control register.

**VOCABULARY**

A program counter (PC) is a CPU register in the computer processor which has the address of the next instruction to be executed from memory.

1. **Decode the instructions into binary (Instruction Decode, ID).**

All programs to be executed are translated into Assembly instructions. Assembly code must be decoded into binary instructions, which are understandable to your CPU. This step is called decoding.

1. **Execute action and move to next step or calculate address (EXE).**

While executing instructions, the CPU can do one of three things: Do calculations with its ALU, move data from one memory location to another, or jump to a different address.

1. **Access memory operand (MEM).**

The CPU must give feedback after executing an instruction, and the output data is written to the memory.

1. **Write back result to register (WB).**

If we need to store the result in the destination location, it is done during the writeback stage, and the register file is updated

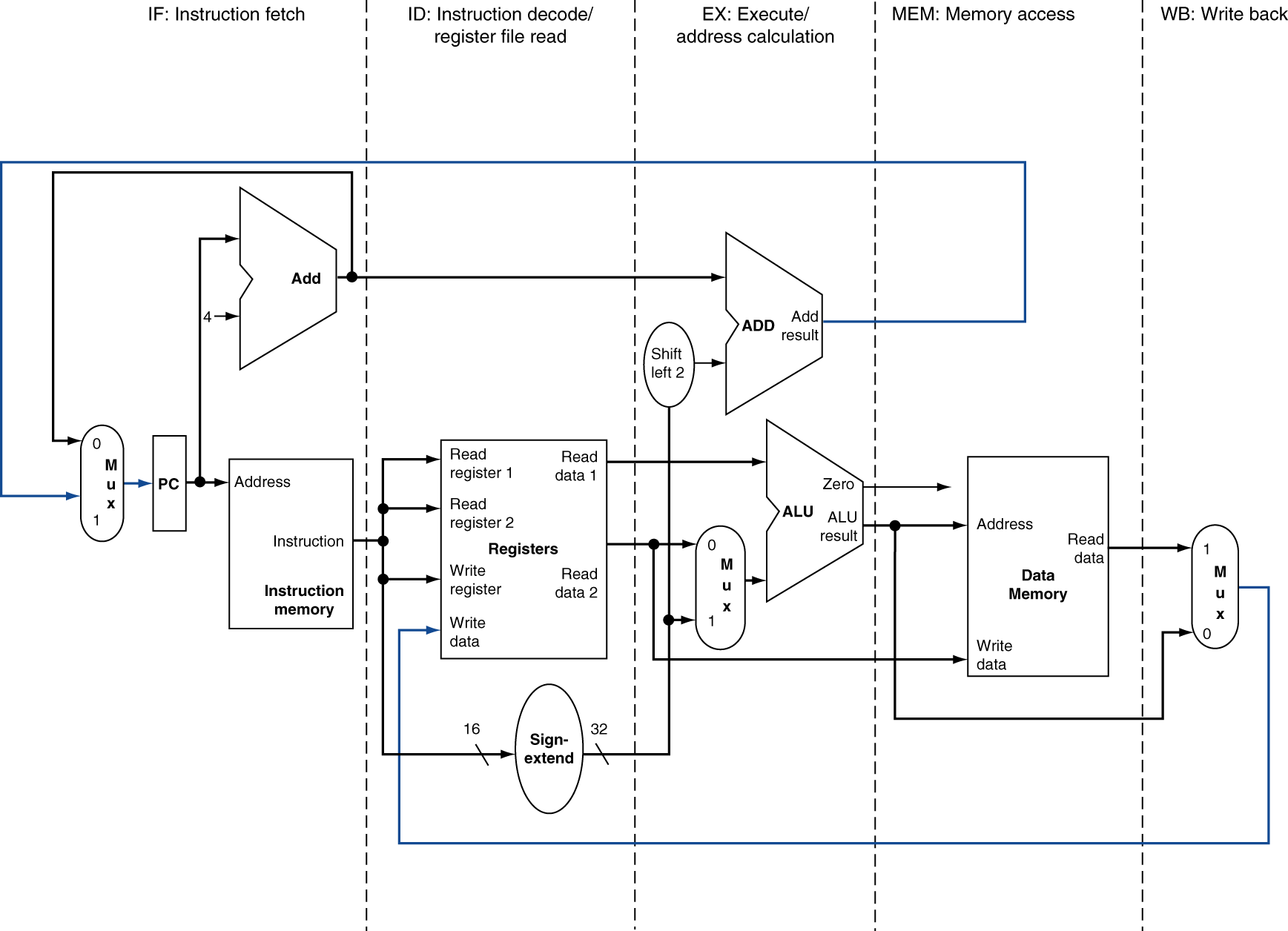


Figure 1.2: 5 stages of processing

Each instruction must be completed in one cycle. Logically, instruction execute one at a time but physically instructions execute in parallel through a technique called pipelining. Pipelining is an implementation technique where multiple instructions get overlapped during execution. Pipelining enhances computer throughput.

A CPU's speed determines how many operations it can carry out. The rate at which one operation is completed in a second is measured in hertz. A computer's speed is typically expressed in gigahertz. The CPU can complete one million simple jobs at 1 GHz speed. The simplest actions a processor can take are considered "simple tasks."

The more instructions your CPU can process in a second, the faster it is, but do not be deceived by this figure. The performance of your computer is impacted by a variety of factors, not only the CPU's speed. To obtain independent results, numerous additional variables, like CPU architecture, cache capacity, and bus speed, must be considered.

The central processing unit (CPU) has three components:

* Control Unit
* Arithmetic Logic Unit
* Registers

**Control Unit**

The control unit reads instructions from memory, decodes them, and then puts them into action. The control unit serves as a middleman, decoding the instructions provided to the processor, directing the other units, such as the Arithmetic Logic Unit (below), using control signals, and sending the processed data back to memory.

**Arithmetic Logic Unit (ALU)**

Digital circuitry inside the processor called an arithmetic logic unit (ALU) handles arithmetic and logical operations by loading data from input registers. The ALU completes the required operations by connecting many transistors after receiving the necessary instructions from the control unit, and then it records the outcomes in an output register. The system clock, memory, secondary storage, and data and address buses are all necessary for the CPU to operate effectively. Some of the arithmetic operations include addition subtraction, multiplications, and divisions. The result of arithmetic computations is put into the accumulator.

**Register** – saves the most frequently used instructions and data.  Processor registers are a common name for the registers used by the CPU. An instruction, a storage address, or any other data, including bit sequences or single characters, may be stored in a processor register.

The different types of registers are explained in detail in section 1.1.7.

**Central Processing Unit**

There are 6 types of central processing units Single Core, Dual Core, Quad Core, Hexa Core , Octa Core, and Deca Core.

Single Core CPU- consists of a single CPU and can execute one instruction at a time.- An example is the Intel 4004 released in 1971.

Dual Core CPU- Consists of two cpu’s that act like one CPU meaning the CPU can multitask e.g Pentium D released in 2005.

Quad-Core CPU-A quad-core CPU has **four cores on a single CPU processor**. The CPU is the greatest choice for multitasking since it equally distributes the workload among its cores. An example of quad core CPU is Athlon II X4 which was released in 2009.

Some computers utilize two or more processors. These consist of separate physical microprocessors located side by side on the same board or on separate boards. Each CPU has an independent interface, separate cache, and individual paths to the system front-side bus. As you will notice from Figure 1.3, the name depicts the number of cpus e.g. Hexa core has 6, octa core has 8 and deca core has 10 cpu’s.



Figure 1. 3: Types of CPU's

Intel and AMD are the two key manufacturers of CPU’s.  These manufacturers are both great, though they tend to be used in different situations.

**Measuring Performance of a Computer**

Computer system performance depends on both the underlying architecture and its implementation. In order to get maximum performance, it is clear that both architecture and and implementation must be optimised.

What do we mean when we say one computer is faster than another? A desktop computer user may claim that a computer is faster when a program runs faster, whereas an Amazon.com administrator may claim that a computer is faster when it completes more transactions per hour. The computer user wishes to reduce response time, also known as execution time, which is the time between the start and completion of an event. A warehouse-scale computer operator may be interested in increasing throughput—the total amount of work completed in a given time.

We frequently want to compare the performance of two different computers, say, X and Y, when comparing design alternatives. The phrase "X is faster than Y" refers to the fact that the response time or execution time on X is faster than on Y for the given task. "X is n times faster than Y" will specifically mean:

Execution timeY/ Execution timeX= n

Example: time taken to run a program

Lets say we have two machines: Machine A takes 10s to complete a task., machine B takes 15s to complete the same task

Execution TimeB / Execution TimeA  
= 15s / 10s = 1.5

So machine A is 1.5 times faster than machine B.

There are many units for quantifying computer performance and many techniques for measuring it. Here are some of the ways of measuring computer performance:

* CPU performance
* Memory System performance
* Input/Output performance
* System Benchmarks

A benchmark is a program or a set of programs chosen to be representative of certain type of task.

### Describe different types of CPUs about use, and power

Section 1.1.3 outlined the types of processors based on number of cores. Processors can also be categorised based on:

* Data bus width e.g., 32-bit and 64-bit processors
* Brand name and manufactures e.g., Intel and AMD
* Architecture- e.g., Reduced Instruction Set Computing RISC and Complex Instruction Set Computing (CISC)
* General purpose processors based on architecture- in this category we have 5 types which are explained below

**Key Difference between RISC and CISC processor**

* In RISC, the instruction set is reduced, and most of these instructions are very primitive, while in CISC, the instruction set is very large that can be used for complex operations.
* RISC computer’s execution time is very less, whereas CISC computer’s execution time is very high.
* RISC code expansion may create a problem, while CISC code expansion is not a problem.
* In RISC, the decoding of instructions is simple, whereas, in CISC, the decoding of instructions is complex.
* RISC doesn’t require external memory for calculations, but CISC requires external memory for calculations.
* RISC has multiple registers sets present, while CISC has only a single register set.

There are five types of general-purpose processors and these are: Microcontroller, Microprocessor, Embedded Processor, DSP and Media Processor.

**Microcontroller**

A microcontroller is basically a type of computer that comes in different packages and sizes. Feedback on reading input and output is the main function of this Microcontroller. It is also commonly referred to as General Purpose Input Output (GPIO). Figure



Figure 1. 4: Example of a microcontroller (  
PIC32MX Series Microcontrollers)

**Microprocessor**

The general-purpose processors are represented by the microprocessor in embedded systems. There are different varieties of microprocessors available in the market from different companies. The microprocessor is also a general-purpose processor that consists of a control unit, ALU, a bunch of registers also called scratchpad registers, control registers and status registers.



Figure 1.5:Example of Microprocessor

Microprocessors are cheap, offer high speed, small in size, consumes less power and generate less heat.

Microprocessors uses a clock signal to control the rate at which instructions are executed, synchronize other internal components and to control the data transfer between them.

**Embedded Processor**

An embedded processor is one type of processor which is designed to control mechanical functions and electrical functions. It consists of several blocks such as the processor, timer, an interrupt controller, program memory and data memory, power supply, reset and clock oscillator circuits, system application-specific circuits, ports and interfacing circuits. Embedded processors are designed especially for coping with the wishes of an embedded gadget.

Embedded processors may be observed in transportable devices like virtual watches, PDAS, digital cameras, GPS gadgets and MP3 players. They also can be located in large structures such as visitor’s lights, systems controlling energy vegetation and factory controllers.

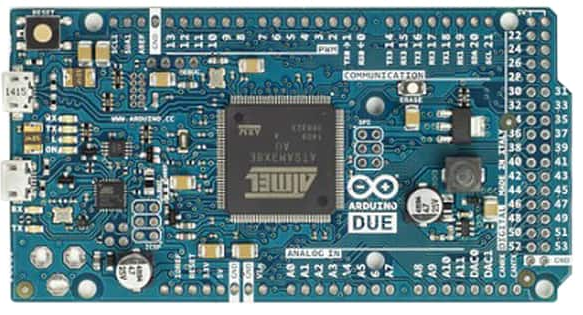


Figure 1. 6:Embedded processors

**Digital Signal Processor**

The digital signal processor is one type of processor used for measuring, filtering and/or compress digital or analog signals. Digital Signal Processors (DSP) take real-world signals like voice, audio, video, temperature, pressure, or position that have been digitized and then mathematically manipulate them. A DSP is designed for performing mathematical functions like "add", "subtract", "multiply" and "divide" very quickly.

The signal processing means analysis and manipulation of signal and can be done via computer or Application Specific Integrated Circuits (ASIC), Field Programmable Gate Array (FPGA) or Digital Signal Processor (DSP) to obtain the clear signal. The DSP processors are used in an oscilloscope, barcode scanners, mobile phones, printers, etc.

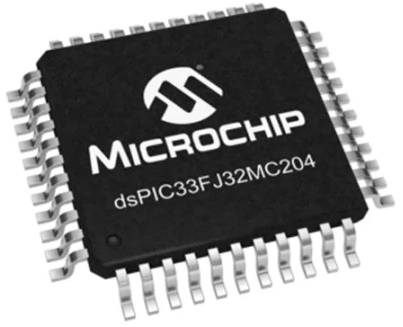


Figure 1. 7: Example of 16bit Digital Signal Processor 40 Million instructions per second (MIPS)

### Define the term computer memory

**VOCABULORY**

Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored.

The memory is divided into large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64k words, then this memory unit has 64 \* 1024 = 65536 memory locations. The address of these locations varies from 0 to 65535. The CPU is responsible for selecting memory cells to read or write data.

The concept of memory and storage can be easily conflated as the same concept; however, there are some distinct and important differences. Put succinctly, memory is primary memory, while storage is secondary memory. We will explain these terms in detail in section 1.1.7.

### 1.1.6 Primary purpose of memory

Memory is central to a computer's operation because it forms the critical link between software and the CPU. The basic function of computer memory is essentially to store data. Depending on the type of data it stores and the role it plays in computer operation, however, memory performs several different functions. Although all of these functions involve data storage, RAM, ROM, flash memory and hard drives each perform a different and necessary function to keep a computer and its peripherals working. Computer memory likewise decides the size and number of programs that can be run all the while, and helps to optimize the capabilities of capable microprocessors

Once the data stored in computer memory, the data will remain there forever. Each time memory is full, then the data can be deleted in part or in whole to be replaced with new data. The performance of a computer depends on memory and CPU. CPU cannot store programs or a large set of data permanently. They are only capable of storing basic instructions required to operate the computer. Therefore, it is mandatory to have the memory to run a computer system properly.

### 1.1.7 The different types of memory and their purpose

There are basically 2 broad categories of memory: Internal and External memory



Figure 1. 8:Categories of Computer Memory

To further assist us in understanding computer memory, we will use a computer hierarchy chart and then explain each of these types.

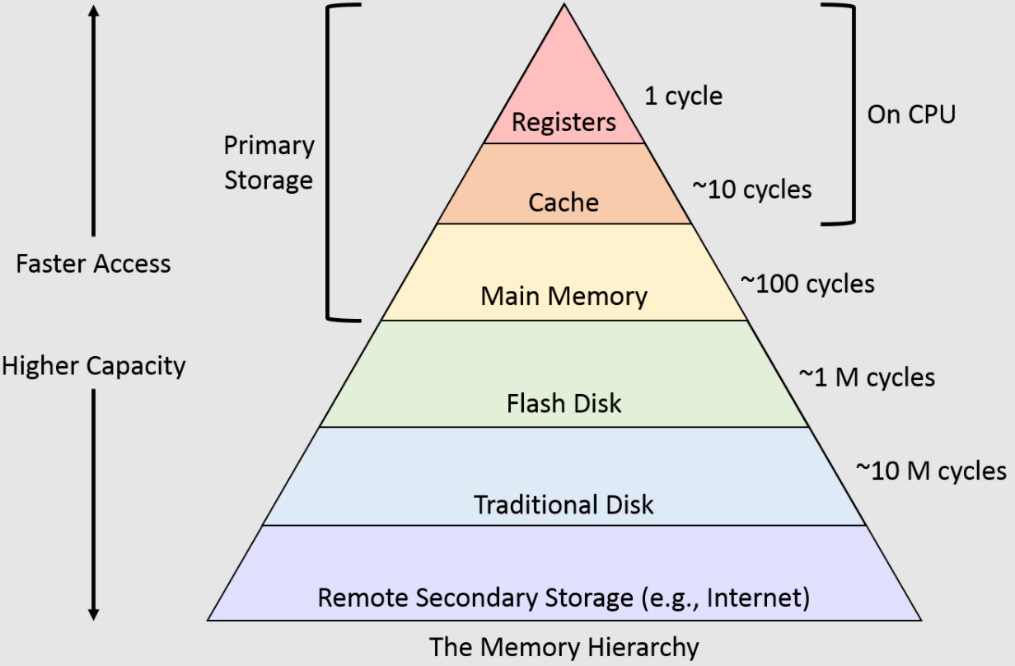


Figure 1. 9: Memory Hierarchy

In computer architecture, the memory hierarchy is an improvement of computer storage into a hierarchy-based modal on response time. It affects performance in a computer algorithm, predictions, architectural design, and lower-level programming constructs involving the locality of reference. Figure 1.9 illustrates an abstraction of memory hierarchy in computers.

**Registers**

Register memory is the smallest and fastest memory in a computer. A register temporarily holds frequently used data, instructions, and memory address that are to be used by CPU. They hold instructions that are currently processed by the CPU. All data is required to pass through registers before it can be processed.  Some of the widely used registers include Accumulator or AC, Data Register or DR, the Address Register or AR, Program Counter (PC), I/O Address Register, and these are explained in Table 1.1.

|  |  |
| --- | --- |
| **Type of register** | **Uses** |
| Data register | stores data, which is being transmitted to or received from a peripheral device. |
| Program Counter | It holds the address of the memory location of the next instruction to be fetched. |
| **Instructor Register** | hold instruction codes and passes them to the Control Unit to be decoded. |
| **Accumulator Register** | is a type of register for short-term, intermediate storage of arithmetic and logic data in a computer's central processing unit (CPU). |
| Address Register | It is a 12-bit register that stores the address of a memory location where instructions or data is stored in the memory. |

Table 1. 1: Common registers

**Cache Memory**

Cache memory is a high-speed memory, which is small but faster than the main memory (RAM). The CPU can access it more quickly than the primary memory. So, it is used to synchronize with high-speed CPU and to improve its performance.

When CPU needs the data, first, it looks inside the L1 cache. If it does not find anything in L1, it looks inside the L2 cache. If again, it does not find the data in L2 cache, it looks into the L3 cache. If data is found in the cache memory, then it is known as a cache hit. On the contrary, if data is not found inside the cache, it is called a cache miss or miss penalty. The solution to reducing cache miss is through implementing several or multilevel caches. **Multilevel Caches** is one of the techniques to improve cache performance by reducing the *“MISS PENALTY”*.

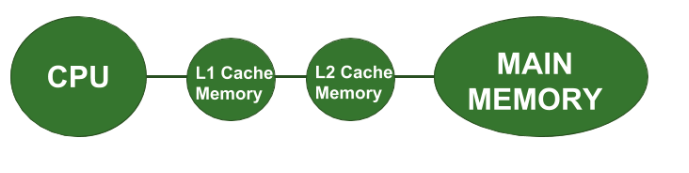


Figure 1. 10: Multilevel caches

**Primary Memory**

Primary Memory is of two types: RAM and ROM.

**Random Access Memory (RAM)**

It is a volatile memory meaning it does not store data or instructions permanently. When you switch on the computer the data and instructions from the hard disk are stored in RAM.

**Advantages of RAM**

1. It is a faster type of memory in a computer.
2. It requires less power to operate.
3. Program loads much faster
4. More RAM increases the performance of a system and can multitask.
5. Perform read and write operations.
6. The processor can read information faster than a hard disc, floppy, USB, etc.

**Disadvantages of RAM**

1. Less RAM reduces the speed and performance of a computer.
2. Due to volatile, it requires electricity to preserve the data.
3. It is expensive than ROM
4. It is unreliable as compared to ROM
5. The Size of RAM is limited.

There are two types of RAM : Static RAM SRAM) and Dynamic RAM (DRAM)

|  |  |  |
| --- | --- | --- |
| Parameter | SRAM | DRAM |
| Read & write | Faster | Slower than SRAM |
| Storage component | Uses transistor to store single bit of information | Uses separate capacitor to store each bit of data |
| Price | Expensive than DRAM | Cheaper than SRAM |
| Power consumption | More | Less |
| Uses | Cache memory | Main memory |
| Density | Less dense | Highly dense |
| Storage per bit | Can store many bits per chip | Cannot store many bits per chip |

**Read-Only Memory (ROM)**

ROM is a memory device or storage medium that is used to **permanently** store information inside a chip. It is a read-only memory that can only read stored information, data or programs, but we cannot write or modify anything. A ROM contains some important instructions or program data that are required to start or boot a computer. It is a **non-volatile** memory.

There are five types of Read Only Memory:

**MROM (Masked Read Only Memory):-** program or data is pre-configured by the integrated circuit manufacture at the time of manufacturing. Its permanent.

**PROM (Programmable Read Only Memory):** It is a type of digital read-only memory, in which the user can write any type of information or program only once.

**EPROM (Erasable and Programmable Read Only Memory)-** stored data can be erased and re-programmed only once in the EPROM memory.

**EEPROM (Electrically Erasable and Programmable Read Only Memory)-** **A high voltage electrical charge is used to erase recorded data from EEROM.**

**Flash ROM: data or instructions** can be written or programmed in small units called block or sector.

**Advantages of ROM**

1. It is a non-volatile memory
2. It is static, so it does not require constant refreshing every time.
3. These cannot be changed accidently
4. It is cheaper than RAM.
5. Simple and reliable

**Disadvantages of ROM**

1. Store data cannot be updated or modify except to read the existing data.
2. It is a slower memory than RAM to access the stored data.

**CMOS- complementary metal-oxide semiconductor**

Alternatively referred to as a RTC (real-time clock), NVRAM (non-volatile RAM) or CMOS RAM, CMOS is short for complementary metal-oxide semiconductor. CMOS is an onboard, battery powered semiconductor chip inside computers that stores information. This information ranges from the system time and date to system hardware settings for your computer.

**Which devices use CMOS?**

* Microprocessors
* Microcontrollers
* Digital logic circuits
* SRAM (Static RAM)

**Secondary Memory**

The secondary storage devices which are built into the computer or connected to the computer are known as a secondary memory of the computer. It is also known as external memory or auxiliary storage. The secondary memory is accessed indirectly via input/output operations. It is non-volatile and cannot be accessed directly by the CPU.

**Examples of secondary memory**

1. Hard Disk Drives
2. Solid State Drive
3. Pen Drive/USB flash drive
4. Secure Digital Card -SD Cards
5. CD/ DVDs

**Cloud Storage**

A service paradigm known as "cloud storage" involves sending and storing data on remote storage systems, where it is then maintained, managed, backed up, and made accessible to users across a network, most often the internet. Users often pay a monthly, per-consumption fee for the storage of their cloud data. Some of the providers are AWS.

### 1.1.8 Explain how data is stored on memory

Data is first transformed into straightforward numbers (Binary Digits-Bits) that a computer can easily store. Bit is the basic unit of memory. At a time, it can be either on or off. Generally, bits are represented using electrical voltage. Voltage presence indicates that the bit is in ON state. Voltage absence indicates that the bit is in OFF state. Here, OFF state is considered as 0. ON state is considered as 1. Computer memory is the collection of several bits. Group of 8 bits are called byte. Second, circuitry within the computer records the numbers. Third, software or programs are used to organize, move, and manipulate the numbers.

**Memory Units**

The smallest unit is a bit

I Byte=8 bits

1 Kilobyte=1024 bytes

1 Megabyte=1024 Kilobytes

1 GigaByte=1024 MegaBytes

1 TeraByte=1024 GigaBytes

### 1.1.9 Define the purpose of the motherboard and its components

A motherboard, often referred to as an mboard, mainboard, base board, system board, planar board, or main circuit board, is the main board and the building block of a computer. A logic board is what it is known as on Apple computers. The motherboard serves as a single platform to connect all of the parts of a computer together. It connects the CPU, memory, hard drives, optical drives, video card, sound card, and other ports and expansion cards directly or via cables. The motherboard of the computer houses the CPU, RAM expansion slots, ROM, USB ports, and PCI slots. It enables communication between the RAM, CPU, and every other piece of hardware. Figure 1.11 shows an example of a motherboard.



Figure 1. 11: i7 Gigabyte Q87M Motherboard

Apart from the other components- the motherboard has a Northbridge and Southbridge

**Northbridge:** An integrated circuit in the motherboard's chipset is in charge of establishing a connection between the AGP, CPU interface, and memory. It has direct connections to the CPU interface, AGP, and memory, unlike southbridge. Northbridge's main responsibility is to provide bus-based communication between the CPU and external devices.

**Southbridge:** This motherboard component is an integrated circuit with a single purpose for which it was produced. I/O controllers, hard drive controllers, and integrated hardware all depend on it.

Some of the popular manufacturers of motherboards includes Intel, ASUS, ABIT, BioStar, Gigabyte and MSI.

**Functions of the Motherboard**

**The functions of a computer motherboard are as follows:**

1. The motherboard acts as the central backbone of a computer on which other modular parts are installed such as the CPU, RAM and hard disks.
2. The motherboard also acts as the platform on which various expansion slots are available to install other devices / interfaces.
3. The motherboard is also responsible to distribute power to the various components of the computer.
4. They are also used in the coordination of the various devices in the computer and maintain an interface among them.

Below are the SEVEN different types of Motherboards:

* **AT & Baby AT**
* **Advanced Technology eXtended ATX**
* **BTX (Balanced Technology Extended):**
* **DTX Discontinuation Transmission**
* **LPX (Low Profile eXtension)**
* **microATX**
* **NLX-  New Low Profile Extended**

### 1.1.10 Describe different types of input hardware based on use and classification

Hardware- Hardware (sometimes abbreviated to HW) can be defined as the physical components that a computer system needs to function.

Software-   consists of written, machine-readable instructions or [programs](https://www.techtarget.com/searchsoftwarequality/definition/program) that tell physical components what to do and when to execute the instructions.

Hardware can be split into input devices, processing devices, output devices and storage devices.

Input devices are used to get data and instructions into the computer. Processing devices are used in the processing of data once it gets into the computer. Input devices can further be split into direct and indirect input hardware.

Direct data entry devices are specific purpose devices designed to automate or speed up the entry of data into the system by minimising human data entry. They have a wide range of uses including in education, retail and in business. They consist of either specialist hardware, software (or both) and come in several different forms. Examples includes:

Magnetic stripe reader-– used to read data from magnetic stripes on mostly banking cards, membership cards or hotel door cards. The stripe on the cards holds data such as membership information.



Figure 1. 12: Magnetic Stripe Reader

Chip readers-read data from the chip on bank cards or shopping cards. The chip and pin reader works by inserting the card into a slot and then entering a PIN (personal identification number)

A picture containing person, indoor, remote, game

Description automatically generated

Figure 1. 13: Chip reader

PIN pads- use to enter data into Automated Teller Machines (ATM), EFTPOS system, entry doors and handheld devices.

Optical mark reader- used to read and input information from a form made in pen or pencil. Typically used to read multiple choice questions. The OMR shines on to the form and less light is reflected where a pencil mark has been made.



Figure 1. 14: Optical Mark Reader

Barcode reader- used to scan codes directly from the products, books and membership cards. Figure 1.14 shows a picture of a Barcode Reader.



Figure 1. 15: Barcode Reader

If the data is in human readable form, it must be converted into machine readable form so that a computer can process it. This process of data conversion is time consuming and error prone that causes a major bottleneck in data processing. Some examples of indirect input devices are: keyboard, mouse and joystick. When you press any key on keyboard, it converts that character into series of electronic pulses and sends to CPU.

**Exercise-Pair exercise**

Categorise the following devices into direct and indirect input hardware.

* Mouse
* Joy Stick
* Light pen
* Track Ball
* Scanner
* Graphic Tablet
* Microphone
* Magnetic Ink Card Reader(MICR)
* Optical Character Reader(OCR)
* Bar Code Reader
* Optical Mark Reader(OMR)

### 1.1.11 How data is transferred between memory and the CPU

The connections between the CPU and Memory are shown in Figure 1.16.

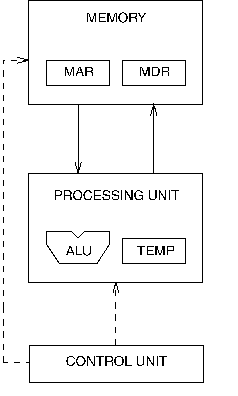


Figure 1. 16:CPU interaction with memory

Communication between memory and processing unit consists of two registers:

* **Memory Address Register (MAR).**

The Memory Data Register (MDR) keeps the data which is transferred between the Memory and the CPU.

* **Memory Data Register (MDR).**

The Memory Address Register (MAR) holds the memory location of data that needs to be accessed.

Reading Data

1. The address of the location is put in MAR.
2. The memory is enabled for a read.
3. The value is put in MDR by the memory.

To write,

* 1. The address of the location is put in MAR.
  2. The data is put in MDR.
  3. The **Write Enable** signal is asserted.
  4. The value in MDR is written to the location specified.
* Buses are the means by which data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory. Address bus Carries the addresses of data (but not the data) between the processor and memory.
* Data bus carries data between the processor, the memory unit, and the input/output devices.
* Control bus carries control signals/commands from the CPU (and status signals from other devices) to control and coordinate all the activities within the computer.

### 1.1.12 Describe the Flow/transfer of data between components

The pursuit of the fastest speed and throughput has been one of the key goals in the development of computer architecture. By utilizing technological potential in the design of the computer components and by giving the computer an appropriate structure and organization, this goal was and is still achieved in two ways. Due to rapid technological advancement in the manufacture of integrated circuits, where component speed and density are continuously rising while costs are falling, the advancement of computer components is quite rapid.

**VOCABULARY**

Data Bus- In computer terminology, a bus is a communication system that allows the transfer of data between components within a computer, or between separate computers.

Computer bus can be in the form of wired cables or electrical wires embedded in the computer motherboard PCB (Printed Circuit Board).

The function of a data bus is to either allow these components to communicate with each other or with the outside world. A data bus can transfer data to and from the memory of a computer, or into or out of the central processing unit (CPU) that acts as the device's "engine." A data bus can also transfer information between two computers. The amount of data that can be transferred by a data bus is referred to as **bandwidth.** If we consider the speed of bus or bit transfer, then one wire or bus transfers millions of bits per second.

Most modern computers today use both parallel and serial buses. • Parallel data buses: o Carry data on many wires simultaneously o Each wire carries one bit of data o Most common parallel buses: Advanced Technology Attachment (ATA), PC card, Small Computer System Interface (SCSI) • Serial data buses: o Has one wire that carries all the bits of data o Most common serial data buses: Universal Serial Bus (USB), FireWire, Serial ATA, Serial Attached SCSI. Figures 1.17 and 1.18 illustrates a data bus and a computer bus respectively.



Figure 1. 17: Data Bus

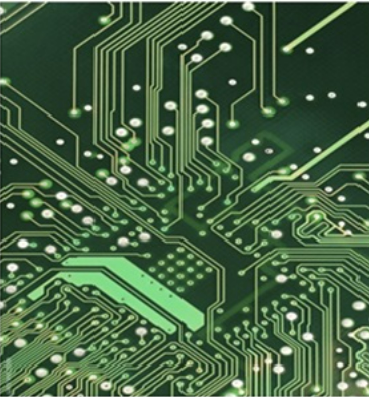


Figure 1. 18: Computer Bus

Apart from data bus, there is also an address bus and a control bus.

* The address bus is used to transfer address bits to the memory.
* The control bus is used to transfer control bits from control units to other components of the computer.

**Data transfer from Universal Serial Bus**

When the software requires data transfer to occur between itself and the USB, it sends a block of data called an *I/O Request Packet (IRP)* to the appropriate pipe, and the software is later notified when this request is completed successfully or terminated by error.

The actual data is sent across the bus in packets. Each *packet* is a bundle of data along with information concerning the source, destination and length of the data, and also error detection information.

Each packet is made up of a set of components called *fields* including the following, summarised in Table 1.2 :

Table 1. 2: A typical data packet. numbers represent size of field in bits, unless otherwise indicated.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sync (8) | PID(8) | Address | Endpoint | Data (0-123 bytes) |

* An eight bit "*SYNC*" synchronisation field used by inputs to correct their timing for accepting data. Marks the start of a data packet.
* The 8 bit Packet Identifier (*PID*) which uses 4 bits to determine the type, and hence format, of the packet data. The remaining 4 bits are a 1's complement of this, acting as check bits. Part of this field determines which of the four groups (token, data, handshake, and special) that the packet belongs to, and specifies an input, output or setup instruction.
* An *address field* which gives the address of the function on the end of the pipe to be used
* The 4 bit *endpoint field*, giving the appropriate endpoint which sends or receives the packet.
* A *data field* consisting of 0-1023 bytes

### 1.1.13 Describe the factors to consider when choosing an input device.

Input devices are peripherals from which computers receive data. Keyboards, mice, scanners, and webcams are some examples. If you select the right accessories, you can extend the capabilities of your computer and increase productivity. Make the most of your next computer accessory purchase by understanding the factors to consider when selecting input devices. Here are some of the factors to consider:

1. **User Needs**- this is the urgency of use of the device in the computer room by users also whether it will satisfy the needs of the user.
2. **Initial cost-**the amount it can cost when buying/purchasing the devices
3. **Maintenance Cost**-the amount that can be used to maintain the servicing of these devises should be considered.
4. **Mode Of Transmission**- how will you transport them to the computer room should be considered
5. **Compatibility With Available Hardware**- will the devices fit to other devices in the room already. A device that fits your needs but doesn't work with your computer is useless.
6. **User- Friendliness**- whether the devices will be used to solve problems and easy to be used by users
7. **Wireless vs wired connection**-Cables is typically faster than a Wi-Fi connection, and it offers other advantages as well. Cable connection is more secure and stable than Wi-Fi. You can test your computer's speeds on Wi-Fi versus an Ethernet connection easily.

### 1.1.14 Describe different types of output hardware

**VOCABULARY**

Any peripheral that accepts data from a computer and prints, projects, or reproduces it is known as an output device.

Output devices are used to provide results of the computer to the users. Examples includes **Monitor, Printer, Projectors and Plotters**. The output may be audio, video, hard copy – printed paper, etc. Output devices convert the computer data to human understandable form. We give input to the computer using input devices and the computer performs operations on the data and displays the output to the user using the output device.

**Monitor** - A computer’s principal output device is a monitor, often known as a Visual Display Unit (VDU). It displays the processed data like text, images, videos, audios, etc. There are two types of monitor viewing screens:

* Cathode-Ray Tube (CRT)
* Display on a Flat Panel Monitor with a Cathode-Ray Tube (CRT)
* Plasma Monitor

**Printer**- it is an output device that creates a hard copy of the processed data or information. Printers are divided into two categories:

* Impact Printer
* Non-Impact Printers

1. **Impact Printers**- In impact printers, characters are printed on the ribbon, which is then smashed on the paper. Here, the characters are printed on to the paper by striking an ink ribbon against it with a hammer or print head. Impact printers are relatively cheap making them ideal for large scale printing. Most of these are used in cooperates. Common examples of impact printers include Dot matrix printer, Daisy wheel printer, Line printer and Chain printer.
2. **Non-Impact Printers:**These printers print characters without the use of a ribbon. These printers are fast because they print one full page at a time producing quality printout. The common examples are laser and inkjet printers.

**Plotter**

A plotter is a [printer](https://www.techtarget.com/whatis/definition/printer) that interprets commands from a computer to make line drawings on paper with one or more automated pens. Some of the examples of plotters include drum plotters, flatbed plotters, electrostatic plotters, and inkjet plotters.

**Projector**

A projector is an output device that reproduces images by projecting them onto a screen, wall, or other surface using images created by a computer or Blu-ray player.

### 1.1.14 Describe the purpose and use of devices such as docking stations for mobile and laptop computers.

A docking station is a cradle for a portable media player that serves to charge and connect the unit to a receiving device. Docking stations enable users with a [laptop computer](https://www.computerhope.com/jargon/l/laptop.htm) to convert it into a [desktop computer](https://www.computerhope.com/jargon/d/desktopc.htm) when at the office or at home. For example, a business user could use a laptop on the road to create a document. When they return to the office, they could attach the laptop to the docking station to use their monitor, speakers, and office [printer](https://www.computerhope.com/jargon/p/printer.htm). Docking stations are now commonly used by developers in increasing the number of displays to reduce number of times of switching through tabs if they are to use one display. Figure 1.19 illustrates an example of a docking station.



Figure 1. 19: Ultra-slim Docking station

# FORMATIVE ASSESSMENT 1.1 INDIVIDUAL TASK

1.1.1 Define the term system unit? (2)

1.1.2 What is the purpose of CPU? (2)

1.1.3 Identify the FIVE stages of information processing cycle. (5)

1.1.4 The rate at which one operation is completed in a second is measured in \_\_\_\_\_\_\_\_. (1)

1.1.5 Identify **THREE** main components of the CPU. (3)

1.1.6 List **THREE** differences between Reduced Instruction Set Computing (RISC) and Complex Instruction Set Computing (CISC). (6)

* + 1. List and describe the FOUR key functions of a computer. (8)

1.1.7 Define memory hierarchy? (2)

1.1.8 Identify the two types of RAM. (2)

1.1.9 What do you understand by the term secondary memory. (2)

1.1.10 List THREE examples each of input, processing and output hardware devices. (9)

**Total :42 Marks**

## 1.2 Purpose of software

**Introduction**

In section 1.1 our discussion mainly centred on hardware components. We must remember that for computers and devices to function, there must be instructions or programs controlling them. Without software, the hardware will not work as expected. We defined software as consisting of written, machine-readable instructions or [programs](https://www.techtarget.com/searchsoftwarequality/definition/program) that tell physical components what to do and when to execute the instructions. In this section we are going to dive deeper into the different software categories and their purposes.

### 1.2.1 Describe the term software

**VOCABULARY**

Software – set of instructions, data or programs used to operate computers and execute specific tasks.

Software can be thought of as the variable part of a computer, while hardware is the invariable part. The two main categories of software are [application](https://www.techtarget.com/searchsoftwarequality/definition/application) software and [system software](https://www.techtarget.com/whatis/definition/system-software). Application software is software that fulfils a specific need or performs tasks. System software is designed to run a computer's hardware and provides a platform for applications to run on. Figure 1.20 shows a diagram outlining computer software.

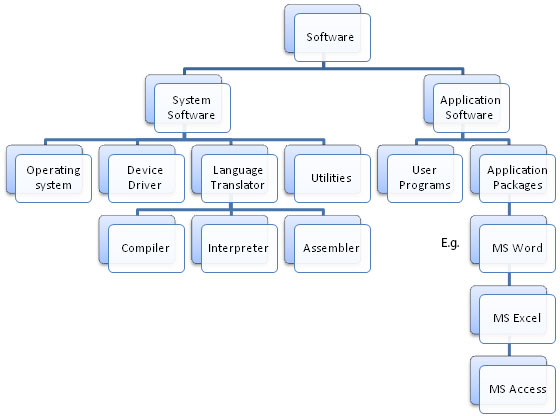


Figure 1. 20:Classification of software

**System Software**

These software packages are made to run the hardware and application software on a computer. The actions and features of the hardware and software are coordinated by the system software. Additionally, it manages how the computer hardware functions and offers a setting or platform in which all other software can operate.

**Operating Software**-An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs conveniently and efficiently. Key functions are file management, memory management, resource management, security management among others. The three basic types of operating systems are stand-alone, server, and embedded.

1. Stand-alone operating system- This is an operating system that runs on a desktop computer, a laptop computer, or a handheld computer is called a stand-alone operating system. Examples are Microsoft Windows, Mac Os, and Linux.
2. Server operating system- these are specifically designed to support the network. Examples includes include windows server, UNIX, Solaris, and Netware.
3. Embedded Operating System-An embedded operating system uses on mobile devices and a wide variety of consumer electronics. It resides on a ROM chip. Common examples are windows mobile Plam OS, iPhone OS, BlackBerry, Google Android, Windows mobile, and Symbian OS.

**Device Drivers**-this software is often considered a type of system software. Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks.

**Utility software**- This software analyses and maintain a computer. This software is focused on how OS works on that basis it performs task to enable smooth functioning of computer. Examples of utility software includes antivirus, backup software, uninstaller, screen saver, file compression, virus scanner, file manager, disk compression tool all are utility software. Some of the utility software come along with operating software for example windows defender, disk cleanup, disk optimisation tools.

**Language Translators**-Translators are computer programs that translates program written in each programming language into a functionally equivalent program in a different language. There are mainly three Types of translators which are used to translate different programming languages into machine equivalent code: **Compiler, Interpreter, Assembler.**

1. **Compiler** is a computer program that translates code written in a high-level language to a low-level language, object/machine code. Examples of languages using compilers are C, C++, C#, Java VB etc.
2. An interpreter program executes other programs directly, running through program code and executing it line-by-line. Examples of interpreted languages are Ruby, Perl, Python, PHP etc. Interpreters are written for multiple platforms; this means code written once can be run immediately on different systems without having to recompile for each.
3. **Assembler**-An assembler translates assembly language into machine code. An assembler takes basic computer instructions and changes them into a design of bits that the computer processor can utilize to perform its fundamental operations.

For now, let us summarise the three translators based on conversion, error detection, execution time, scanning as shown in Table 1.3

Table 1. 3: Comparative analysis of translators

|  |  |  |  |
| --- | --- | --- | --- |
| Parameters | Compiler | Interpreter | Assembler |
| Conversion | High level to Machine Code or Binary code | High level to Machine language | Assembly language to Machine language |
| Scanning | Entire code before converting to binary code | Translates line by line | Converts source code into object code and then to machine code |
| Error Detection | Error report after a full scan | Detects error line by line | Detects error in the first phase before second phase can start |
| Execution time | Less time compared to interpreters | Takes more time than compilers | Takes more time than compilers |

**Application Software**

Application software consists of application packages and user programs. Applications packages are always bought as a suite of programs for instance, Microsoft Office, LibreOffice and they provide common end-user needs. Examples of application programs include word processors e.g. Microsoft Word, WordPad, LibreOffice Writer, presentation programs e.g., PowerPoint, databases e.g. Microsoft Access, LibreOffice Base etc. User programs could be any example of programming languages such as C, C#, C++, Java, Kotlin, Go, R, Perl etc. In chapter 4, we are going to dive deeper into the C++ compiler.

### 1.2.2 Describe the purpose and function of software

**Functions of computer software**

* **Software** can make the computer compare data, make logical decisions, do mathematical calculations, store and retrieve data and instructions from primary or secondary sources, and carry out sequences of tasks.
* The software enables computer users to obtain what they need from the computer.
* **Software makes** the computer work towards giving the outputs in the manner the user wants them to, such as output on a screen, printouts, sounds, sending emails, etc.
* **Software** is used to translate programs written in different languages into machine languages, e.g., compilers, interpreters, and assemblers.

### 1.2.3 Basic concepts of software.

Operating systems serve as interfaces between users and computer hardware. Figure 1.21 illustrates how the operating systems interacts with hardware and providing interface for users. Interfaces are separated into two categories: command-line and graphical.

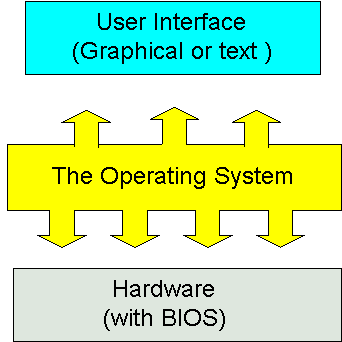


Figure 1. 21: Interfacing for users

* **Command-line Interface**: In a command-line interface, users interact with the computer through commands. Command line interface is difficult to work with as it requires knowledge of command language.
* **Graphical User Interface (GUI):** As a graphical user interface does not need you to memorize the command language, it is easier to understand. Users interact with the computer through icons and menus.

### 1.2.4 System software vs application software

Table 1. 4:compares system software and application software.

|  |  |
| --- | --- |
| System Software | Application Software |
| System software is mainly designed for managing system resources. | Application software is designed to accomplish specific tasks. |
| Programming of system software is complex. | Programming of application software is comparatively easy. |
| A computer cannot run without system software. | A computer can run without application software |
| System software do not depend on application software | Application software depend on system software and cannot run without system software. |
| System software are typically written or programmed in a low-level language such as machine language or assembly language. | Application software are mostly written in high level languages such as Java, C++ etc. |

Figure 1.2.2 illustrates an overview of computer hardware and software.

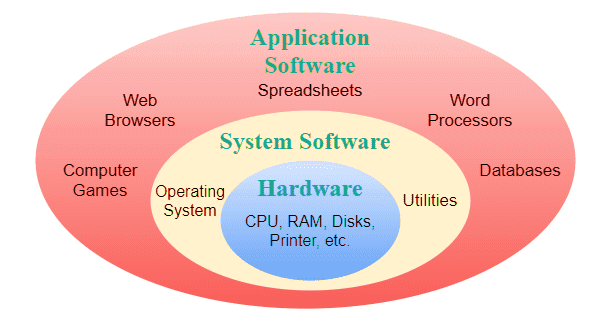


Figure 1. 22: Overview of computer

### 1.2.5 Differentiate between: Shareware, Freeware, Open-Source Software and Proprietary software, Firmware

Computer software can also be classified according to availability and shareability and the categories are freeware, shareware, open source closed source and midlleware. Figure 1.23 illustrates the different categories of software.

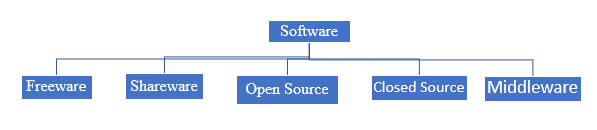


Figure 1. 23: Categories of software

**Freeware**

Freeware is the software that is available to use for free of cost without any limitations. Example- Some of the google software is freeware.

**Shareware**

The software is copyrighted and distributed for free only for testing purposes. After the trial period ends, you must pay. For example, Microsoft Office (if you download MS-Office from the Microsoft website they give a one-month free trial after that you must pay for it yearly or monthly).

**Open Source**

This is provided for use, modification, and redistribution. Open-source software is downloaded from the internet at no cost. Example- Google play store apps, etc.

**Middleware**- is software that is used to bridge the gap between applications and other tools or databases. Some examples of middleware activities include handling data and [API](https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API) management, authentication and messaging services.

**Closed source software**

Closed source software is software for which the source code is not freely available.  It is developed and provided to the user as a fully compiled, executable set of files. The developer often provides support to users after purchase and ensures that the software works as expected. This is usually developed if you cannot find the specific software you want off the shelf and you approach software development houses to custom make for your specific needs.

**Middleware**

Middleware is software that bridges gaps between other applications, tools, and databases to provide unified services to users. It is commonly characterized as the glue that connects different software platforms and devices together.

**Proprietary**

Proprietary software is computer software where the source codes are publicly not available only the company that has created can modify it. A good example is Microsoft products where users must get a licence key to install them on to their devices.

**Firmware**

Firmware is a type of software that is embedded directly in a piece of hardware to make the hardware work as intended. Firmware is programmed by the manufacturer and is installed on a digital device right in the factory. All computing devices have firmware.

### 1.2.6 Discuss the process of how software is obtained and installed

Computer software can be installed from a compact disk or any other portable devices. Software which comes loaded on a compact disc is autoplay allowing to start the installation process immediately. If not, you will need to load the storage device and locate the installation file with .exe extension. Another alternative would be to install from the command line. In this case, you will need to navigate to the folder containing the installation file and run the executable setup file. Many times, this can be done by typing setup or install at the prompt to start the installation.

Nowadays, the most common way to get new software is to download it from the Internet. software development houses are allowing customers to access software from their cloud storage after payment. Once a payment is received the user will receive a key which allows them to install the software. Applications like Microsoft Office and Adobe Photoshop can now be purchased and downloaded right to your computer. You can also install free software this way. For example, if you wanted to install the Google Chrome web browser you just download it from their server and install it to your personal computer. The installation file will be saved to your computer in .exe format.

### 1.2.7 Differentiate between online software and installed software.

**Installed software** is also referred to as desktop software. Installed software must be purchased in physical form (such as a CD) or downloaded from the internet. In either case, the program is installed using an installation program. Once the software is installed, it is ready to use.

**Benefits of installed software**

1. Installed software can be viewed when offline
2. Installed software is accessible on any device irrespective of connectivity
3. Installed software offers a better User eXperience
4. User has full control of the data

**Online software** is any program accessed over an Internet connection using a Web browser. The website pages act as a user interface. Online software run on a secure, external server, which also stores any data collected.

**Benefits of online software**

1. No software to install
2. Complete flexibility on device type
3. No maintenance required.
4. Connect from anywhere at any time with internet
5. Centralised storage offering safe and secure data

### Discuss the following terms in relation to software

**VOCABULARY**

Compatibility is the capacity for two systems to work together without having to be altered to do so.

Compatibility issues come up when users are using the same type of software for a task, such as word processors, that cannot communicate with each other. This could be due to a difference in their versions or because they are made by different companies. There are two types of compatibility: forward compatibility and backward compatibility.

Products that are designed to be compatible with future versions of themselves are referred to as **forward compatible**. Products designed for compatibility with older versions are said to be [**backward compatible**](https://www.techtarget.com/whatis/definition/backward-compatible-backward-compatibility).

Any application which can allow users to revert or rollback to the previous version is said to be backward compatibility.

**Software Version** means a unique build of the software released by a company identified by the alpha-numerical sequence assigned to it together with a corresponding release date e.g. [version 5](https://www.lawinsider.com/dictionary/software-version).9.1 released on 13 October 2022.

**Software Patches**

A software patch is a program that makes changes to software installed on a computer. Software companies issue patches to fix bugs or security problems in their programs or add new functions to the software.

**Service Packs**

A service pack (SP) is a collection of updates and fixes for an operating system or a software program. SPs are usually made up of patches. Many of these patches are often released before a larger SP, but the SP allows for an easy, single installation.

Both system software and application software serve to facilitate user interaction with computer systems and the accomplishment of various activities. There are variations in their designs and intended uses, though. Application software is created to address specific user demands in order to carry out specific tasks, as opposed to system software, which is created to manage system resources or processes.

# FORMATIVE ASSESSMENT 1.2 INDIVIDUAL TASK

1.2.1 Define the term software. (2)

1.2.2 Identify THREE types of language translators. (3)

1.2.3 Differentiate a compiler from an interpreter. (4)

1.2.4 Define the following terms:

* Freeware
* Middleware
* Shareware
* Open Source (8)

1.2.5 Identify the FOUR categories of system software.

1.2.6 Discus FOUR benefits of online software (10)

1.2.7 Define the term compatibility with regards to software. (2)

**Total: 29 Marks**

## 1.3 The Linux shell

The shell, to put it simply, is a program that receives keyboard commands and sends them to the operating system for processing. A program called bash serves as the shell on the majority of Linux computers. You can communicate with the shell by using the terminal emulator, which launches a window. Linux employs a wide range of different terminal emulators.  Among them are gnome-terminal, konsole, xterm, rxvt, kvt, nxterm, and eterm, among others. On Raspberry Pi, terminal we can run Linux commands.

### 1.3.1 Launch a new Linux terminal on the Raspberry Pi

To launch the terminal on Raspberry Pi, you will need to navigate to the Raspberry Icon on the Desktop. Click Accessories and select terminal. Figure 1.24 illustrates how the Raspberry terminal looks like.

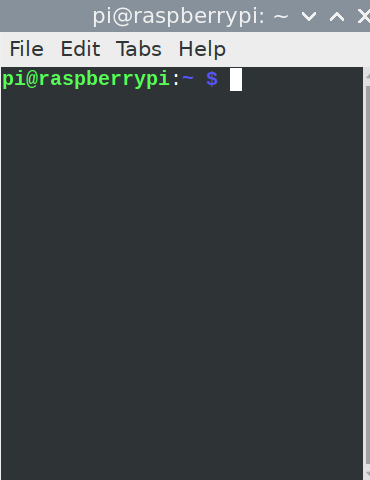


Figure 1. 24: Raspberry Terminal

### 1.3.2 Use the man command to get help

**man** is the system's manual pager. Each page argument given to man is normally the name of a program, utility or function. In Linux, man is an interface to view the system's reference manual. A user can request to display a man page by simply typing man followed by a space and then argument. Here its argument can be a command, utility or function. A manual page associated with each of these arguments is displayed. The syntax of man command is as follows:

$ man -options argument

For example:

man - -usage

Figure 1.25 shows the output of man –usage command to print usage syntax.

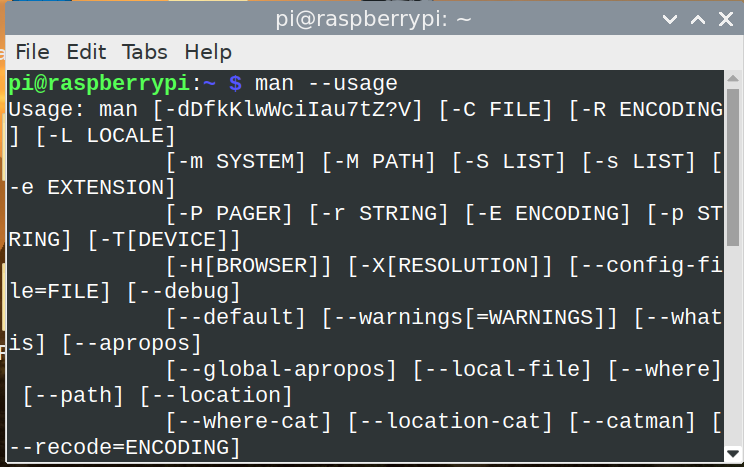


Figure 1. 25: Using man command

Another example could be bringing up the help by using man -h command as illustrated in Figure 1.26.



Figure 1. 26:using man -h command

So, to avoid buying a lot of books and saving trees- man is your answer. It provides the documentation at your fingertips.

### 1.3.3 Expand a Linux file path and explain each element

A file path is the human-readable representation of a file or folder’s location on a computer system. Files and folders on Linux are given names containing the usual components like the letters, numbers, and other characters on a keyboard. But when a file is inside a folder, or a folder is inside another folder, the / character shows the relationship between them. Figure 1.3.4 illustrates the file structure of the file add2.py. Just like in Windows where the file structure starts with C: on Raspberry Pi home is the root directory. If, for instance you want to see the exact file path, navigate to it on the terminal and type in the command: readlink -f <filename> as shown in Figure 1.27, we managed to find out the full path of the file called add2.py which is in /home/pi/Desktop folder

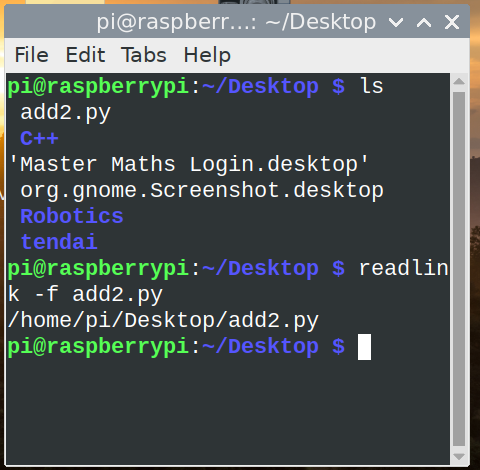


Figure 1. 27:File structure

Looking at Figure 1.27, home is similar to the Windows directory. Pi is a sub directory. Desktop is the current folder where the file add2.py is located.

### 1.3.4 List the contents of the current folder using the ls command

To list contents of the directory, the ls commands does the trick. See Figure 1.27

We listed all the files and folder on the Desktop through ls command.

### 1.3.5 Change directly location using the cd command

To navigate from one directory to another, Linux uses cd (change directory) just like in Windows. Figure 1.3.5 illustrates how we used the cd command to navigate from Desktop folder to C++ folder.

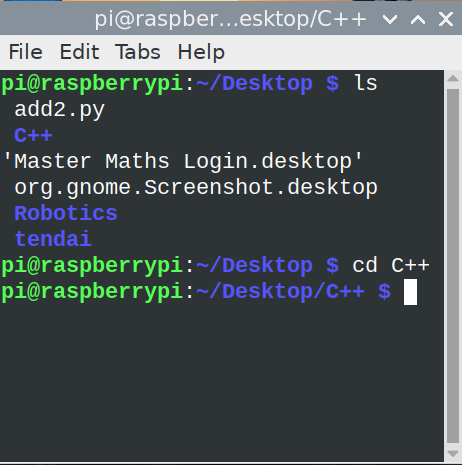


Figure 1. 28:Using cd command

If you are in a specific directory, you can navigate straight to the root directory by typing cd /. cd .. will make you move one level less from the current directory.

### 1.3.6 Create a new folder using the mkdir command

**mkdir** command in Linux allows the user to create directories as shown in Figure 1.29. The user running this command must have sufficient permissions to create a directory in the parent directory; otherwise, a "permission denied" warning may appear.

**Syntax**

mkdir [options...] [directories ...]

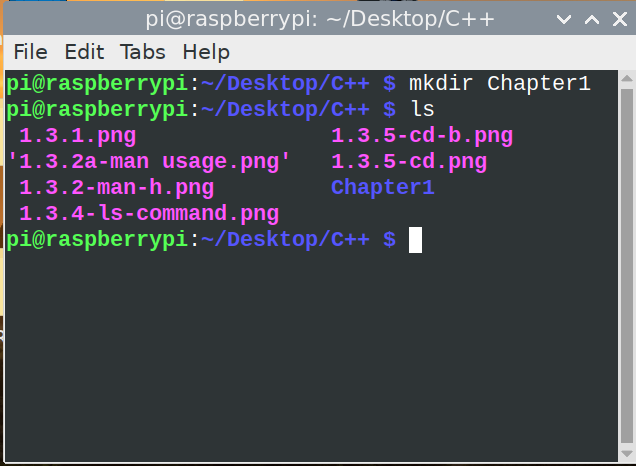


Figure 1. 29:Using mkdir command

In Figure 1.29, we managed to create a directory called Chapter1 inside the C++ folder. If you want to create two folders at once, just type the mkdir command and folder names and leave space between the names. There are other options which can be used together with mkdir command such as:

**- -v or - -verbose**: It displays a message for every directory created.   
**- -help**: It displays the help related information and exits.

### 1.3.7 Remove a folder using the rmdir command

rmdir command is used remove empty directories from the filesystem in Linux. The rmdir command removes each directory specified in the command line only if these directories are empty. Figure 1.30 shows how we removed the directory called Chapter1 from the C++ directory. After listing all files using the ls command, we can see that the directory is not there anymore.

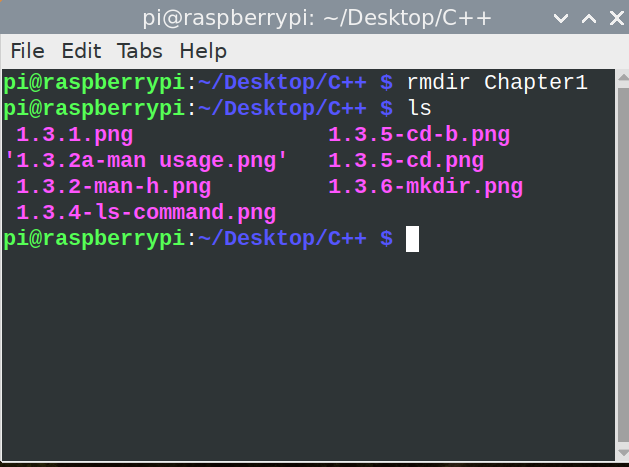


Figure 1. 30:Using rmdir command

### 1.3.8 Remove a file using the rm command

The rm command removes the entries for a specified file, group of files, or certain selected files from a list within a directory. When you use the rm command, user confirmation, read permission, and write permission are not required before a file is removed. In Figure 1.31, we demonstrate how to remove the file called newfile.py from C++ directory. We also demonstrated how to create a new file using touch command.

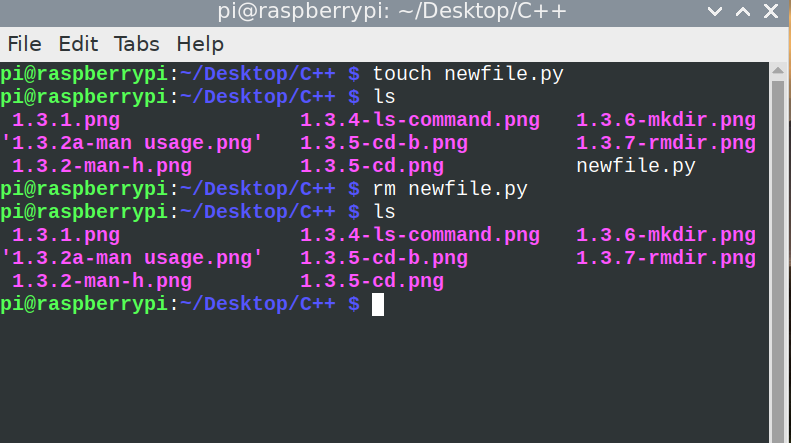


Figure 1. 31:Using rm command

### 1.3.9 Rename a file using the mv command

Use the mv command to change the name of a file without moving it to another directory. To rename a file, type the following:

$ mv <currentfilename> <proposedfilename>

Figure 1.32 illustrates how we renamed the file called newfile.py to newfile1.py by executing the following command:

$ mv newfile.py newfile1.py

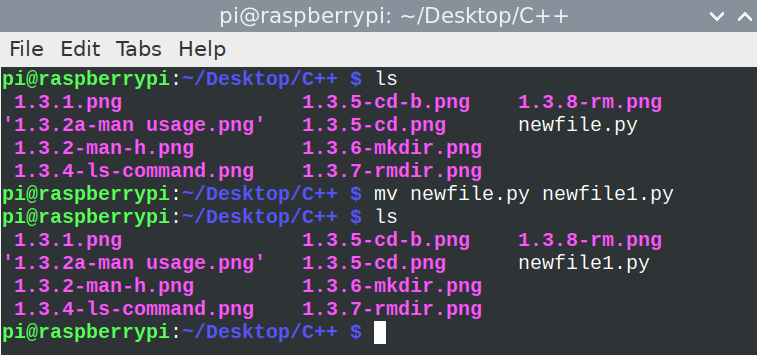


Figure 1. 32:Changing filename using mv command

### 1.3.10 Copy a file using the cp command

Use the [cp](https://www.ibm.com/docs/en/ssw_aix_72/c_commands/cp.html) command to create a copy of the contents of the file or directory specified by the source file or source directory parameters into the file or directory specified by the TargetFile or TargetDirectory parameters. If the file specified as the TargetFile exists, the copy writes over the original contents of the file without warning. If you are copying more than one SourceFile, the target must be a directory. Figure 1.33 demonstrates the cp command used to copy newfile.py to C++ folder.

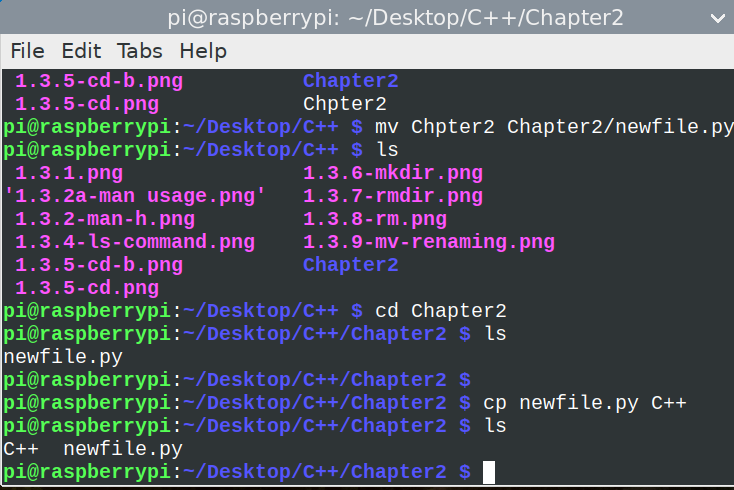


Figure 1. 33:Using cp command

### 1.3.11 Clear the command prompts screen using the cls command

Clear command clears the text on the terminal. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen. clear ignores any command-line parameters that may be present. Cls is used on Windows platform. The equivalent on Linux is clear.

### Run an executable file from the command line

To run an executable file the user rights of those files must be set correct. This can be done by doing the following:

Syntax

chmod +x <filename>.

chmod +x sample.sh.

There are more Linux commands to perform different operations. The link below will give a guide of other Linux commands.

**e-Link**

<https://www.guru99.com/linux-commands-cheat-sheet.html>

# FORMATIVE ASSESSMENT 1.3 INDIVIDUAL TASK

1.3.1 Explain how to launch a Linux terminal on Raspberry Pi. (3)

1.3.2 What command is used to list all files and directories in a file path. (1)

1.3.3 Use Raspberry terminal commands **ONLY** to do the following:

* Create a folder called terminal\_commands on the Raspberry Pi desktop using a terminal command.
* Navigate to the created directory using commands.
* Enter the code to show the path of the current directory
* Create a file called summative.docx

Copy all the commands in sequence of execution which you used to do all the above steps. (6)

1.3.4 Navigate to the terminal\_commands folder created in question 1.3.3.

Add the following line of text into the file called summative.docx:

* “Hello fellow programmers”.
* Write the full command which you used to add the line. (3)

1.3.5 Create another file called formative.docx using terminal commands in the same directory as the one created in question 1.3.3.

Add the following text to the formative.docx : “Terminal commands are easy”

Merge the contents of the two files together into a file called final.docx.

Read the contents of the final.docx file from the terminal.

Write all the steps of achieving this through the terminal

The final output must be as follows:

Hello fellow programmers

Terminal commands are easy

(8)

1.3.6 You are supposed to complete the table 1.3 below by pairing the correct terminal command with its function.

Table 1. 5: Terminal commands

|  |  |
| --- | --- |
| **Command** | **Function** |
| ls | This command is used to delete files within a directory. |
| createdir | To permanently delete an empty directory. |
| rm | Is used to know more about a command and how to use it |
| pw | Used to create a new directory |
| rmdir |  |
| mkdir |  |
| man |  |

**Total: 25 Marks**

**SUMMATIVE ASSESSMENT INDIVIDUAL TASK**

# SUMMATIVE ASSESSMENT 1.4 INDIVIDUAL TASK

* + 1. Differentiate a Memory Buffer Register from a Memory Address Register. (4)

1.4.2 Differentiate SRAM and DRAM. (4)

1.4.3 List and describe **FIVE** factors to consider when choosing an input device. (10)

1.4.4 Complete the following abbreviations with respect to read Only Memory.

1. MROM
2. PROM
3. EPROM
4. EEPROM  **(4)**

**1.4.5 Identify two differences between a microcontroller and a microprocessor. (6)**

1.4.6 Define the term proprietary software. (2)

1.4.7 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is a design principle in which a program or piece of hardware is designed to work with new software or devices in the future. (1)

1.4.8  If a file is removed in Linux using ‘rm’ then it can be recovered by the user. True /False. (1)

1.4.9 Explain what happens when the following command is entered on the terminal:

$ echo "The process id is" $$$$ (2)

1.4.10 Create a folder called Practicals on the Desktop using terminal commands on your Raspberry Pi.

Add the following statement into a file called next.txt”

"It is the control unit that determines which machine instruction is to be executed next\_file.txt"

Add another line.

“ The control unit is the brains of the computer”

Read the text in the next\_file.txt to display on the terminal without exiting from the current folder.

Your output should be as follows:



Display the current path of the file.

Write all the terminal commands.

(6)

**Total: 40 Marks**

**Practical Activity: Software Installation**

Search for Anydesk and install the software on your machine. You can go to this link: <https://anydesk.com/en/downloads/thank-you?dv=win_exe>

Share the pin numbers of the software with your friend and try controlling each other’s machine.

Discuss with your friend what the software does.

## References

William, S., 2010. Computer organization and architecture designing for performance. EIGHTH EDITION.

Hennessy, J.L. & Patterson, D.A., 2011. *Computer architecture: a quantitative approach*. Elsevier.

Dumas II, J.D., 2018. *Computer architecture: Fundamentals and principles of computer design*. CRC Press.

# Module 2 Problem solving in computer programming

After you have completed this module, you should be able to :

* Define the term problem solving
* Define the term computational thinking
* Describe the phases of the PLDC (Program Development Life Cycle)
* Describe the purpose of problem solving leading to solutions
* Explain and apply various problem-solving steps.
* State in own words Clarity on what needs to be done
* What is known or given?
* What is missing or needed?
* Devise a plan/algorithm (storyboard – visual or
* textual)
* Look for patterns
* Look at related problems, known solutions
* Examine simpler or special cases
* Make a table, create diagram, use guess and check,
* work backwards, identify sub-goal
* Carry out the plan/implement the algorithm (write
* the code)
* Look back/test (see if it works)
* Check results against original problem. Does it make sense? Is there another solution?)
* Use appropriate tools and techniques to present a solution. Range:
* User stories (written by the client and provide the requirements)
* Noun-verb analysis of user stories
* List of nouns provides identification of objects and state
* List of verbs provides identification of behaviour
* Acceptance tests (does the program meet the
* requirements?)
* Define the term algorithm and its purpose in the problem-solving process. (Range: Basic concepts of an algorithm.
* What is an algorithm? Develop a clear understanding of the problem presented.)
* Implement and understand the basic algorithmic constructs used to create a flowchart. Range: Input, Output, Processing and Calculations, Selection, Iteration
* Create a flowchart to present a particular algorithm and its associated tasks
* Interpret a basic flow chart and describe its intended operation / function

**Introduction**

The fundamental goal of computer science is problem solving. The problems that we want to solve can come from any real-world problem or perhaps even from the abstract world. We need to have a standard systematic approach to solving problems. First, programmers must comprehend how humans solve problems, then they must comprehend how to transform this "algorithm" into something that a machine can perform, and finally they must comprehend how to "code" the syntax (needed by a computer) to accomplish the task. Sometimes a machine will approach an issue entirely differently than a human would.

## 2.1 Problem solving process and concepts

### 2.1.1 Define the term problem solving

**VOCABULARY**

Problem solving is the sequential process of analysing information related to a given situation and generating appropriate response options.

Using a computer, issues are resolved by getting some sort of user input (such as keyboard/mouse data or gaming control motions), processing the input, and creating some sort of output (e.g., images, text, sound). Hard disks or network devices may be used to store and transmit data occasionally.

### 2.1.2 Define the term computational thinking

**VOCABULARY**

Computational thinking is an interrelated set of skills and practices for solving complex problems, a way to learn topics in many disciplines, and a necessity for fully participating in a computational world.

Using computational thinking, we can break down complex problems into its component parts and come up with potential answers. Then, we may communicate these solutions in a manner that is understandable to a machine, a person, or both.

The four cornerstones of computational thinking

* **decomposition** - breaking down a complex problem or system into smaller, more manageable parts
* **pattern recognition** – looking for similarities among and within problems
* **abstraction** – focusing on the important information only, ignoring irrelevant detail
* **algorithms** - developing a step-by-step solution to the problem, or the rules to follow to solve the problem

A complex problem is broken down into a succession of smaller, more manageable problems using computational thinking (decomposition). Then, each of these smaller issues can be examined separately, considering how comparable issues have been resolved in the past (pattern recognition), concentrating solely on crucial information, and eliminating irrelevant data (abstraction). Next, straightforward guidelines or procedures to address each of the lesser issues can be created (algorithms).

### 2.1.3 Describe the phases of the PLDC (Program Development Life Cycle)

**VOCABULARY**

The program development life cycle (PDLC) is a set of steps or phases which are used to develop a program in any programming language.

PDLC provides an organized plan for breaking down the task of program development into manageable chunks, each of which must be successfully completed before moving on to the next phase. Figure 2.1 illustrates the six phases of program development lifecycle.

Figure 2. 1: Program Development Lifecycle

Program Development Life Cycle consists of six stages, and these are:

1. Requirements Analysis
2. Designing the solution
3. Coding the program
4. Testing and Debugging
5. Implementation of the solution and support
6. Documentation

**Requirements Analysis**

In this phase, we need to understand what the problem statement is, what is our requirement and what is the output expected based on the problem. All these are included in the first phase of program development life cycle.

**Designing the solution**

The first step in program design is to concentrate on the major objective that the program is attempting to accomplish, after which the program is divided into manageable parts, each of which contributes to the overall objective. Top-bottom programming, often known as modular programming, is this method of program design. Different design tools are used in this stage and some of these includes pseudocode, flowcharts, algorithms, Use case diagrams, data flow diagrams, decision tables, Input Processing -Output tables, hierarchy charts and many others. Systems architect and UX/UI designers are the key in this stage of development.

**Coding the program**

The actual programming instructions for the actions specified in the previous phase are now written or implemented using a programming language. In this stage, we write the program. Utilizing programming languages like C, C++, Java, etc., we create the software to address the problem at hand. The technique of programming using only well-defined control structures is known as *Structured programming*. Programmer must follow the language rules, violation of any rule causes *error*. These errors must be eliminated before going to the next step. Developers both front end and back end are the common people at this stage.

**Testing and Debugging**

*Software Testing* is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free.

The application will run following the correction of syntax problems. However, the program's results might not be accurate. This is a result of a logical error in the software. A logical error occurs when the programmer makes a mistake when creating a solution to a problem. Therefore, by carefully reviewing the program output using Test data, the programmer must identify and fix logical problems. Bugs are a collective term for both syntax and logical errors. The process of identifying errors and eliminating them is known as *debugging*.

**Testing levels**

Table 2.1shows the common testing techniques in software development process.

Table 2. 1:Testing levels

|  |  |
| --- | --- |
| **Testing level** | **Activities** |
| Integration testing | This brings together two or more application modules to make sure they work together. Defects in interface, communication, and data flow between modules are also revealed by this kind of testing. |
| Unit testing | Unit testing is typically done throughout the application development process and its goal is to make sure that every single unit or component works as planned. |
| Regression testing | This determines if adding additional features results in a decrease in an application's functionality. |
| Stress testing | This gauges a program's robustness by seeing how much stress it can withstand before failing. |
| White Box Testing | In white-box testing, test cases are created using programming knowledge and an internal viewpoint of the system. Typically, this testing is carried out at the unit level. |
| Black Box Testing | Software testers use the black box testing technique to assess the functionality of the program being tested without examining the internal code layout. |
| Acceptance testing | a testing technique performed to determine whether or not the software system has met the requirement specification |

**Implementation of the solution**

At this testing, all testing has been done and the developers are happy with the product. The program is deployed (installed) at the user’s site. Document the solution using comments with the program, and support documentation for users (i.e., those that won’t be looking at the code).

**Documentation**

The software project is nearly finished after testing. During the design phase, the structure charts, pseudocodes, flowcharts, and decision tables created become documentation for others involved with the software project. This phase concludes with the creation of a manual that includes an overview of the program's functionality, tutorials for beginners, detailed explanations of major program features, reference documentation for all program commands, and a detailed description of the error messages generated by the program.

**Review and Maintenance**

The application is monitored in this instance as well until the user gives it the all-clear. Even after it is finished, the software still needs to be constantly maintained and assessed. The programming team upgrades the software and corrects program flaws during software maintenance.

I hope you have learnt how software is developed, starting with brainstorming to identify the problem up to the last stage of review and maintenance. You must remember that execution of these stages can be done only after completion of the previous phase.

### 2.1.4 Describe the purpose of problem solving leading to solutions

**VOCABULARY**

Problem solving is a mental process that involves discovering and analysing a particular issue, developing strategies, and organizing skills and knowledge to overcome obstacles and find viable solutions that best resolve the problem.

There are 6 steps that you should follow to solve a problem:

1. Understand the Problem

2. Formulate a Model

3. Develop an Algorithm

4. Write the Program

5. Test the Program

6. Evaluate the Solution

As you will see, the above stages are just close to those of program development lifecycle.

Consider a simple example of how the input/process/output works on a simple problem:

Example: Calculate the average grade for all students in a class.

1. Input: get all the grades
2. Process: add them all up and compute the average grade.
3. Output: output the answer to either the monitor, to the printer, to the USB flash drive or hard disk … or a combination of any of these devices.

Let us try to make use of the problem-solving steps to solve the above problem.

**Step 1: Understand the problem**

Some of the key questions that should be answered in order to understand the problem are:

* What input data/information is available?
* What does it represent?
* What format is it in?
* Do I have everything that I need?
* What output information am I trying to produce and in what format?

We are aware that the input in our example is a collection of grades. But we must comprehend the structure of the grades. Each grade could be a letter grade from A+ to F or a number from 0 to 100. If the answer is a number, the grade may be a real number, such as 73, or a full integer.

**STEP 2: Formulate a Model**

We will calculate the average of the incoming grades in our example. Therefore, we must be aware of the procedure (or formula) for calculating the average of a collection of numbers.

Assuming that the input data is a bunch of integers or real numbers x1,x2,…,xn representing a grade percentage, we can use the following computational model:

Average1 = (x1 + x2 + x3 + … + xn) / n

where the result will be a number from 0 to 100.

We can also draw up a chart of grade values and symbols.

**STEP 3: Develop an Algorithm**

To develop an algorithm, we need to represent the instructions in some way that is understandable to a person who is trying to figure out the steps involved. To allow even non-computer people to understand the development, pseudocode would be the best to depict the scenario.

*Pseudocode* is a simple and concise sequence of English-like instructions to solve a problem.

**EXAMPLE 2.1**

Input: Number of terms n

Output: Sum and average of those n terms

Procedure SumAverage

Sum=0

i=0

Repeat for each i <n:

Read a number x

i =i +1

Avg=sum /n

Sum =sum + x

Print sum and average

**STEP 4: Write the Program**

At this stage, the design structure is changed into a programming code of choice. In our case we used C++ programming language.

Here is the sample program

**EXAMPLE 2.2**

#include <iostream>

using namespace std.

 int main () {

    int i, count, sum, inputArray[500];

    float average.

     cout << "Enter number of elements\n";

    cin >> count.

    cout << "Enter " << count << " elements\n”.

    // Read "count" elements from user

    for (i = 0; i < count; i++) {

        cin >> inputArray[i];

    }

    sum = 0;

    // Find sum of all array elements

    for (i = 0; i < count; i++) {

        sum += inputArray[i];

    }

  average = (float)sum / count;

    cout << "Average = " << average;

     return 0;

}

After coding we will need to compile our program to check for errors. *Compiling* is the process of converting a program into instructions that can be understood by the computer.

**STEP 5: Test the Program**

Once a program has been written and has passed compilation, it must be checked to see if it solves the problem it was designed to and that the answers are accurate. If everything is in order, the output from your application should be correct after running the program. Your program should be bug-free as much as possible. Test your program with many test cases (called a test suite) to find bugs effectively.

**STEP 6: Evaluate the Solution**

You should re-consider the original problem and ensure that your answer is formatted into a proper solution after your program produces a correct result. Cross check with the objective of the problem and the results. Now you can deploy the solution and constantly maintain it. Sometimes, the program may misbehave days or months after deployment due various reasons. So, you will need to regularly test it with the acceptable test data.

So now we have illustrated how you can implement problem-solving steps. You should make sure to follow these steps as a guide to coming up with the correct solution to a given problem.

2.1.5 Explain and apply various problem-solving steps.

Polya’s First Principle: Understand the problem. His techniques of problem solving became very common and are used in the modern days. In 1945 George Polya published the book “How to Solve It” which quickly became his most prized publication. In this book he identifies four basic principles of problem solving.

First Principle: **Understand the problem**

* Some of the questions that can help elucidate the problem could be as follows:
* Do you understand all the words used in stating the problem?
* What are you asked to find or show?
* Can you restate the problem in your own words?
* Can you think of a picture or diagram that might help you understand the problem?
* Is there enough information to enable you to find a solution?

Second Principle: **Devise a plan**

There are numerous rational approaches to resolve issues, says Polya. The greatest way to learn how to choose an effective strategy is to solve lots of issues. Following the stages will make problem solving get easier and easier. Some of the strategies include:

* Guess and check
* Eliminating possibilities
* Use a model
* Use direct reasoning
* Look for a pattern
* Draw a picture

Third Principle: **Carry out the plan**

Typically, this stage is simpler than creating the strategy. Given that you have the essential abilities, all you really need is care and patience. Stick to the strategy you've picked. If it doesn't stop failing, throw it away and pick another. Don't be fooled; this is how math is done, even by experts.

Fourth Principle: **Look back**

The idea behind looking back is to allow the process of reflection on what worked well and what did not work. People learn more from reflections. You will be able to foresee what approach to take to address difficulties in the future by doing this.

**eLink**

https://youtu.be/zhL3EMFSm6o

### 2.1.6 Use appropriate tools and techniques to present a solution.

**VOCABULARY**

Agile software development refers to software development methodologies centred around the idea of iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

User stories are among some of the techniques to present a solution. User stories are mostly used in agile development methodologies.

In agile product management and software development a user story is a succinct, casual, and straightforward summary of software functionality that the system's end users need. Its main objective is to offer software features that will meet the needs of the consumer. User stories are simple, yet extremely powerful constructs: they describe pieces of functionality from a user’s point of view, expressed in a solid, compact way.

***Why User Stories?***

1. User stories provide an excellent way to [define your product with clarity](https://www.theinnovationmode.com/the-innovation-blog/the-mvp-recipe-what-to-include-in-your-product-when-and-why).
2. User stories help to achieve *cross-team clarity* on *what* to build, for *whom*, *why,* and when.
3. User stories encourage participation by non-technical members.
4. User stories help in [defining the entire product](https://www.theinnovationmode.com/the-innovation-blog/the-mvp-recipe-what-to-include-in-your-product-when-and-why) — as a set of solid, wisely-prioritized stories.

User stories are completely from the end-user perspective which follows the Role-Feature-Benefit pattern. The pattern is as follows:

As a [ type of user], I want [ an action], so that [ some reason]

Some points outlined which are taken into consideration during writing user stories like

1. Requirements
2. Tasks and their subtasks
3. Actual user
4. Importance to user words/feedback
5. Breaking user stories for larger requirements

A common principle when writing user stories is to consider 3C’s.

* Card-write stories on cards
* Conversation-conduct conversations with the people involved to get more information.
* Confirmation-meet the acceptance criteria of the software.

**Example of a user story**

As a customer I want ability to book a movie ticket that matches my preferences so that I get to quickly and easily book the movie of my choice.

User Story Description

This feature will involve user selecting a specific city, searching for the movie name, selecting a specific timeslot, and then completing the order booking formalities.

**Acceptance Criteria:**

1. User navigates to the search movie page
2. User selects city
3. User enters movie name
4. System searches for the matching movies and displays results
5. User selects specific movie timing and proceeds to book
6. User enters no. of guests and seats
7. User provides payment information
8. System validates the payment information and confirms the booking
9. System sends email/SMS to the user with booking confirmation details

**Key Terms**

* **user story** – one short sentence in everyday language of the end user that states what a user does as part of his or her work
* **acceptance criteria** – features that must be present in the final system for the user to be satisfied
* **use case** – an activity that the system performs, usually in response to a request by a user
* **user goal technique** – a technique to identify use cases by determining what specific goals or objectives must be completed by a user
* **event** – something that occurs at a specific time and place, can be precisely identified, and must be remembered by the system

# FORMATIVE ASSESSMENT 2.1 INDIVIDUAL TASK

2.1.1 Define the term problem solving as applied in programming. (2)

2.1.2 What is meant by the term computational thinking? (2)

2.1.3 Identify **FOUR** cornerstones of computational thinking. (4)

2.1.4 With the aid of a diagram, represent the SIX stages of program development life cycle. (6)

2.1.7 List SIX steps which must be followed in problem solving. (6)

2.1.8 Write an algorithm that reads three numbers and prints the value of the largest number. (10)

2.1.9 Define the term algorithm as used in programming. (2)

**Total: 32 marks**

## 2.2 Construct an algorithm and present a solution to a given problem

Earlier in this chapter, we spoke about the program development lifecycle and the associated stages. We highlighted the design tools, but we want to go a bit deeper with examples. In this section we will discuss flowcharts, algorithms and pseudocodes.

### 2.2.1 Define the term algorithm and its purpose in the problem-solving process.

**VOCABULARY**

An algorithm is a finite sequence of steps expressed for solving a problem.

In mathematics, computer science, and related subjects, an algorithm is a finite sequence of steps expressed for solving a problem. An algorithm can be defined as “a process that performs some sequence of operations in order to solve a given problem”. Algorithms are used for calculation, data processing, and many other fields.

Let’s say that you have a friend arriving at the railway station, and your friend needs to get from the railway station to your house. Here are three different ways (algorithms) that you might give your friend for getting to your home.

**The taxi algorithm:**

* Go to the taxi
* Get in a taxi
* Give the driver my address.

**The call-me algorithm:**

* When your train arrives, call my mobile phone.

**The bus algorithm: Outside the railway station, catch Golden Arrow Bus.**

* Transfer to taxi at Wynberg station.
* Get another taxi to HoutBay.
* Get off by the Shell Garage in Plumstead
* Walk two blocks west to my house in towards Victoria Hospital.
* My house is painted in yellow

Three reasons for using algorithms are:

* **Efficiency**- Certain types of problems, like sorting, occur often in computing. To solve such problems, efficient algorithms must be used, considering the time and cost factors involved in each algorithm.
* **Abstraction**- Algorithms provide a level of abstraction in problem solving. When we see a more complicated problem in a simplified light, we can consider the simpler problem to be an abstraction of the more complicated one.
* **Reusability**- Algorithms are frequently reusable.

Algorithms can be expressed in many different notations, including natural languages, pseudocode, flowcharts and programming languages.

### 2.2.2 Basic algorithmic constructs used to create a flowchart

**VOCABULARY**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

**Characteristics of an Algorithm**

* **Unambiguous**
* **Input-must have 0 or more inputs**
* **Output** − should have 1 or more well-defined outputs.
* **Finiteness**-must terminate after several steps
* **Feasibility**-should be feasible with available resources
* **Independent**-must have step by step directions independent from other programs

Design an algorithm to add two numbers and display the result.

**EXAMPLE 2.3**

Step 1 − START

Step 2 − declare three integers a, b & c

Step 3 − define values of a & b

Step 4 − add values of a & b

Step 5 − store output of step 4 to c

Step 6 − print c

Step 7 − STOP

**Types of Algorithms**

* **Brute Force Algorithm**- A brute force algorithm solves a problem through exhaustion: it goes through all possible choices until a solution is found.
* **Recursive Algorithm**- In this instance, a problem is divided into multiple smaller components and repeatedly called by the same function.
* [**Backtracking Algorithm**](https://www.geeksforgeeks.org/backtracking-algorithms/)**-** Every time a solution fails, we go back to the original problem, build on the new one, and repeat the process until the problem is solved or all potential solutions have been considered.
* **Divide-and-conquer algorithm-**A a problem is repeatedly divided into two or more subproblems of the same or similar type, until these are sufficiently straightforward to be solved by themselves.

**PROBLEM**: Find the sum of 5 numbers. Use an algorithm to solve the problem

**Solution:** Algorithm (in simple English)

**EXAMPLE 2.4**

Step 1: Start

Step 2: Initialize sum = 0 and count = 0    (PROCESS)

Step3: Enter n    (I/O)

Step 4: Find sum + n and assign it to sum and then increment count by 1 (PROCESS)

Step 5: Is count < 5 (DECISION)

if YES go to step 2  
 else  
Step 6: Print sum (I/O)

Step 7: Stop

**Flowcharts**

**VOCABULARY**

A Flowchart is a type of diagram (graphical or symbolic) that represents an algorithm or process.

A flowchart is a schematic representation of an algorithm or a stepwise process, showing the steps as boxes of various kinds, and their order by connecting these with arrows. A process or program can be designed or documented using flowcharts. Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction. A flowchart typically shows the flow of data in a process, detailing the operations/steps in a pictorial format which is easier to understand than reading it in a textual format.

**Flowchart symbols**

Table 2. 2:Flowchart symbols

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Function** |
|  | Start/End | An oval shape represents a start or end of a program |
|  | Arrows | A line is a connector that shows relationships between the representative shapes |
|  | Input/Output | A parallelogram represents an input or output to the program |
|  | Process | A rectangle represents a process |
|  | Decision | A diamond indicates a decision |
|  | Connector symbol | Indicates that the flow continues where a matching symbol has been placed. |

Consider that we need to find the sum, average and product of 3 numbers given by the user.

Algorithm for the given problem is as follows:

**EXAMPLE 2.5**

Step 1: Start

Step 2: Read num1, num2, num3

Step 3: Compute Sum (S) as num1 + num2 + num3

Step 4: Compute Average (A) as S / 3

Step 5: Compute Product (P) as num1 x num2 x num3

Step 6: Display Sum (S), Average (A), Product (P)

Step 7: Stop

Take note that it is not mandatory to start with developing an algorithm and then you draw a flowchart. Design tools are selected based on what the programmer sees as best representing the given scenario. Let us now represent the above algorithm using the flowchart symbols discussed in Table 2.2. Figure 2.2 represents the same process depicted using an algorithm in example 2.5.

Text

Description automatically generated

Figure 2. 2: Example of flowcharting

Let us use the problem which we did earlier on sum of numbers.

**PROBLEM**: Find the sum of 5 numbers.

We already wrote the algorithm. Now we are going to use flowchart symbol to depict the scenario.

**Flowchart for summing up 5 numbers**

:A picture containing diagram

Description automatically generated

Figure 2. 3: Flowchart for sum of 5 numbers

**Basic guidelines for drawing a flowchart**

In drawing a proper flowchart,

* All necessary requirements should be listed out in logical order.
* The flowchart should be neat, clear, and easy to follow.
* There should not be any room for ambiguity in understanding the flowchart.
* The flowchart is to be read left to right or top to bottom.
* A process symbol can have only one flow line coming out of it.

### 2.2.3 Create a flowchart to present a particular algorithm and its associated tasks

In section 2.2.2 we identified the common shapes for making a flowchart. Let us consider the following challenge.

**Problem:**

Add two numbers from the keyboard and display the output. Design a flowchart to solve the problem.

To solve this problem, we will take a variable sum and set it to zero. In the previous example, we did not set values of num1, num2 and num3 to 0. It is always a good practice to set default values for variables. Then we will take the two numbers number1 and number2 as input. Next, we will add both the numbers and save the result in the variable sum i.e., sum = number1+number2. Finally, we will print the value stored in the variable sum.

Here is the algorithm for the above example.

Algorithm (in simple English)

**EXAMPLE 2.5**

* Initialize sum = 0 (PROCESS)
* Enter the numbers (I/O)
* Add them and store the result in sum (PROCESS)
* Print sum (I/O)

**Flowchart**

Text

Description automatically generated

Figure 2. 4: Adding two numbers

Figure 2.4 is a sequential flowchart since steps are followed one after the other in sequence without a choice of condition followed. If the program requires a choice of events based on specific conditions being met, conditional flowcharts will be used. A conditional flowchart is used when a condition is imposed on a problem. The condition will either be true or false. The course of the problem depends on the answer to the condition. Here is the challenge:

**Write a program to check if the given number is a multiple of 3 or not.**

Here are the steps:

1. Input the number 'n'.
2. Divide 'n' be 3.
3. If reminder equals 0, print 'n' is a multiple of 3'.
4. If reminder does not equal to 0, print 'n' is not a multiple of 3'.

Use a flowchart to depict the scenario in the algorithm.

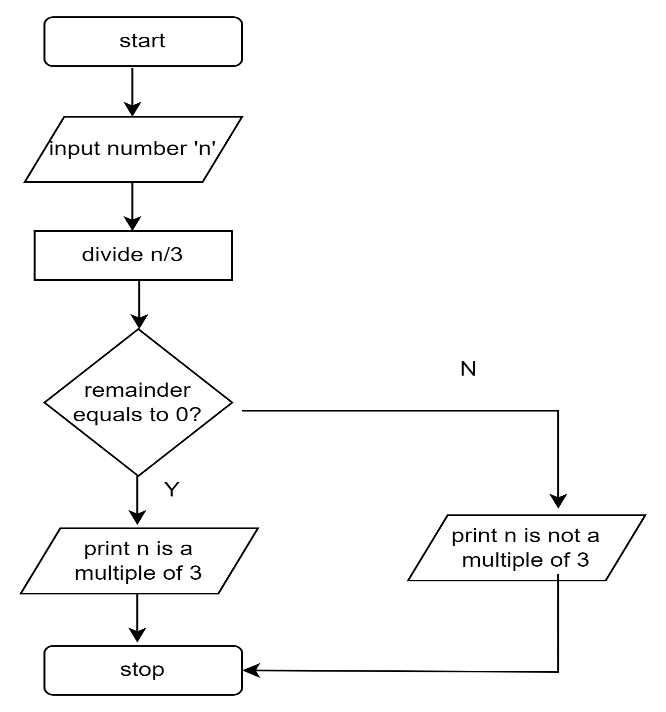


Figure 2. 5: conditional flowchart

**Repetition flowchart** is used when the program requires the act of repeating one or more steps in a process. It involves a **branching***backwards* away from the normal sequence of steps towards an earlier step. The branching decision is based on a **condition** (relationship between the values of known data) at the time that the branching test is performed.

**Program:**

**Write a program to print hello five times.**

In this case, we take a variable count and set it to zero. Then we print "Hello World" and increment count by 1.

i.e., count = count + 1

Next, we check if count is less than 10. If this is TRUE, then we again print "Hello World" and increment the variable count. On the other hand, if the condition if FALSE then we will stop.

Here is the flowchart for the problem.

A picture containing diagram

Description automatically generated

Figure 2. 6:Repetition flowchart

Let us consider the following challenge.

**Challenge:** Draw a flowchart to log in to Facebook account

To log in to Facebook account we first enter the Facebook URL www.facebook.com in our browser. This request is sent to the Facebook server and it responds by sending us the home page of Facebook. Next, we enter our registered Email ID and Password and click the Login button. Then our login credential is checked. If it is correct, we are show our profile. On the other hand, if the login credential is wrong then an error occurs, and we are prompted to re-enter our Email ID and Password.

**Solution**

Figure 2.7 represents the logical steps of logging on to Facebook.

Diagram

Description automatically generated with medium confidence

Figure 2. 7: Logging on to Facebook

**PROBLEM:**

Write an algorithm to determine a student’s final grade from 4 tests scores and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

A picture containing polygon

Description automatically generated

Figure 2. 8:Average of four test scores

### 2.2.4 Interpreting a basic flow chart and its intended operation

How frequently have you found it difficult to comprehend a process after being given a thorough explanation? In these circumstances, flow charts are a helpful tool since they make a process simple to comprehend at a glance. They effectively illustrate what happens at each stage and how this influences other decisions and actions using only a few basic words and symbols.

**When to Use a Flow Chart**

* Defining a process.
* [Standardize](https://www.mindtools.com/pages/article/5s-system.htm)  a process.
* Communicating a process
* Identify bottlenecks or waste in a process

We have already discussed the key shapes symbols used in creating a flowchart. Interpreting your Flowchart will help you to :

* Determine who is involved in the process.
* Form theories about root causes.
* Identify ways to streamline the process.
* Determine how to implement changes to the process.
* Locate cost-added-only steps.
* Provide training on how the process works or should work.

**Benefits of Using Flowcharts**

The benefits of flowcharts are as follows:

* **Communication:** Flowcharts are a better way to communicate the logic of a system to all parties involved.
* **Effective analysis:** A flowchart can help you analyse a problem more effectively.
* **Proper documentation:** Program flowcharts are useful for program documentation, which is required for a variety of reasons.
* **Efficient Coding:** During the systems analysis and program development phases, the flowcharts serve as a guide or blueprint.
* **Proper Debugging:** The flowchart aids in the debugging process.
* **Efficient Program Maintenance:** The use of a flowchart simplifies program maintenance. It allows the programmer to focus his or her efforts more effectively on that aspect.

**Limitations of Using Flowcharts**

Although a flowchart is a very useful tool, there are a few limitations in using flowcharts which are listed below:

* **Complex logic:** The program logic can be quite complicated at times and in such instances flowchart become more complex and clumsy.
* **Alterations and Modifications:** If changes are required, the flowchart may need to be completely redrawn.
* **Reproduction:** Because flowchart symbols cannot be typed, reproduction of flowcharts is difficult.
* The essentials of what is done can easily be lost in the technical details of how it is done.

# FORMATIVE ASSESSMENT 2.2 INDIVIDUAL TASK

2.2.1 Discuss in brief **ANY FOUR** testing phases in program development lifecycle. (12)

2.2.2 Consider the following scenario.

Tickets are sold for a concert at $20 each, if 10 tickets are bought then the discount is 10%, if 20 tickets are bought the discount is 20%. No more than 25 tickets can be bought in a single transaction. Design a flowchart to depict the above scenario. (10)

2.2.3 List SIX characteristics of a good algorithm. (6)

2.2.4 Mr January’s class of programming has 25 students who sat for the test. The teacher asks you to design a flowchart to calculate the average from 25 exam scores. (10)

2.2.5 Write an algorithm to depict the given scenario.

Scenario: Find the area of a Circle of radius r.

**HINT**

Inputs to the algorithm: Radius r of the Circle.

Expected output: Area of the Circle (6)

2.2.6 List and explain SIX benefits of using flowcharts. (12)

**Total : 66 Marks**

# SUMMATIVE ACTIVITY 2.3 INDIVIDUAL TASK

2.3.1 Explain **FOUR** cornerstones of computational thinking. (8)

2.3.2 What is a conditional flowchart? (2)

2.3.3 You are requested to design a program to convert temperature recorded in Fahrenheit into degrees Celsius.

The formula for conversion is:

Celsius=5/9 \* (F-32) (5)

2.3.4 One of the uses of computer programs is mathematical calculation. Here is a problem Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation.

Formula: ax2 + bx+ c = 0

Hint: d = sqrt ( b2-4ac), and the roots are: x1 = (–b + d)/2a and

x2 = (–b – d)/2a

algorithm (7)

flowchart (7)

2.3.5 Consider the following statement.

Problem Statement

Calculate the interest of a bank deposit. You are to read the amount, years and interest rate from the keyboard and print the interest amount.

**Hint: I=Amount \* Years \*Rate/100**

Draw a flowchart to depict the scenario. (7)

2.2.6 List and explain THREE limitations of flowcharts. (6)

* + 1. The symbol denotes \_\_\_\_\_\_\_

1. I/O
2. Flow
3. Process
4. Decision (1)

2.2.8 A box that can represent two different conditions.  
a) Rectangle  
b) Diamond  
c) Circle  
d) Parallelogram (1)

2.2.9 In computer science, algorithm refers to a pictorial representation of a flowchart.  
a) True  
b) False (1)

2.2.10 The operation represented by parallelograms is \_\_\_\_\_\_\_\_.  
a) Input/Output  
b) Assignment  
c) Comparison  
d) Conditions (1)

**Total :46 Marks**

## References

Polya, G., 2004. *How to solve it: A new aspect of mathematical method* (Vol. 85). Princeton university press.

# Module 3:Concepts of programming for single board microprocessor or microcontrollers

**3.1 Introduction to IO on single board computing**

After you have completed this module, you should be able to :

* Expand the term GPIO
* Compare and contrast some of the major advantages of python compared to other programming languages
* Explain and identify where the GPIO pins are located
* Differentiate between a compiler and an interpreter
* Explain the purpose of the GPIO pins
* Discuss the major characteristics of the python programming language as an interpreted one
* Define the term physical computing
* Differentiate between a shell and an IDE
* Read and interpret a Pi GPIO Pin guide
* Define the term physical computing
* Explain what the terms 3V3, 5V, GND GP2 means on the GPIO board
* Discuss how Arduino is used to enable physical computing

**3.2 Visual Programming and solution development**

* Construct (code/write) using the visual tool, debug and run simple programs incorporating: Declaration of variables of different types, use and assignment of values to variables, incorporating program constructs with sequence, selection and iteration structures. Expose and apply various programming concepts as part of the coded solution such as:
  + retrieving remainders: modulus
  + differentiate between real value division and integer division
  + comparison operators and performing logical comparisons
  + incorporate and write code constructs to perform basic calculations such as area, volume, VAT and simple formulae, typical calculations done in other subjects
  + include conditional constructs [if and if then-else] (up to a maximum of two nested levels)
  + Include iteration (looping) structures [fixed counter loop]
  + incorporate a combination of iteration and condition structures as part of the solution (i.e. program code)
* Write code which applies programming language tools and constructs to draw various shapes (turtle type commands) on an output screen/window. Reinforce concepts such as:
  + - Sequence
    - Selection
    - Iteration
    - Creation of objects and shapes
* Design a coding solution to a problem incorporating a combination of different programming constructs which include:
  + Sequence
  + Selection
  + Iteration
* Design and develop solutions for specific problems that include computational thinking and applying software engineering principles.
* Explore lists/arrays (storing and accessing a list of numbers and strings) and containers. (Range: Manipulating lists/arrays such as adding, deleting, replacing, inserting items.)

### 3.1 Introduction to IO on single board computing

**Introduction**

In Level 2, we learnt to program using Python language on a single board computer. We made use of the Raspberry Pi as our single board computing hardware. We then moved to visual programming using Scratch to describe processes through illustrations. In this level, we are still going to use single board computing and our hardware will be Arduino boards and the programming language we are going to use is C++.

Raspberry Pi and Arduino are two very popular examples of single board computing hardware commonly used in among programming and electronics. However, the two boards are quite different. Raspberry Pi serves as a learning tool for computer programming, whereas Arduino is designed for rapid programming and circuit prototyping. There is one major difference between them: Arduino is a microcontroller board, while Raspberry Pi is a microprocessor-based minicomputer. Each board has unique benefits and drawback. We have already learnt the common features of a Raspberry Pi. We will proceed to discus the features of an Arduino board.

**VOCABULARY**

**Single-board computing** is a complete, functioning computer in which the microprocessor, input/output functions, memory, and other features are all built on a single circuit board, with RAM built in at a pre-determined amount and with no expansion slots for peripherals.

### 3.1.1 Exploring the Arduino board

Arduino is open-source electronics prototyping platform that contains both hardware and software founded by Massimo Banzi and David Cuartielles in 2005. So, to be very clear. Arduino consists of hardware and software. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or simplified version of C++ IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. The Arduino does not require a separate piece of hardware to load fresh code onto the board; instead, you can do so by using a USB-B cable, in contrast to most earlier programmable circuit boards. Arduino are the common microcontrollers preferred by most users because they are inexpensive, cross platform, off a simple and clear programming environment and built on open source and extensible hardware and software.

**Types of Arduino boards**

Numerous Arduino boards have been built throughout the years for thousands of projects, ranging from simple everyday objects to complex scientific apparatus. The list of Arduino boards includes the following such as:

* Arduino Uno (R3)
* Arduino Nano
* Arduino Micro
* Arduino Due
* Arduino Mega (R3) Board
* Arduino Robot

The features of different types of the common Arduino boards are listed in Table 3.1.

Table 3. 1: General specifications of common types of Arduino boards

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name of Arduino board** | **Processor type** | **Memory** | **Digital I/O** | **Analogue I/O** |
| Arduino Uno | 16Mhz ATmega328 | 2KB SRAM, 32KB flash | 14 | 6 input, 0 output |
| Due | 84MHz AT91SAM3X8E | 96KB SRAM, 512KB flash | 54 | 12 input, 2 output |
| Mega | 16MHz ATmega2560 | 8KB SRAM, 256KB flash | 54 | 16 input, 0 output |
| Leonardo | 16MHz ATmega32u4 | 2.5KB SRAM, 32KB flash | 20 | 12 input, 0 output |

We are going to discuss the Arduino Uno board as this is one of the common boards among many users.

**Features of Arduino Uno Board**

Figure 3.1 illustrates an Arduino Uno pin structure.

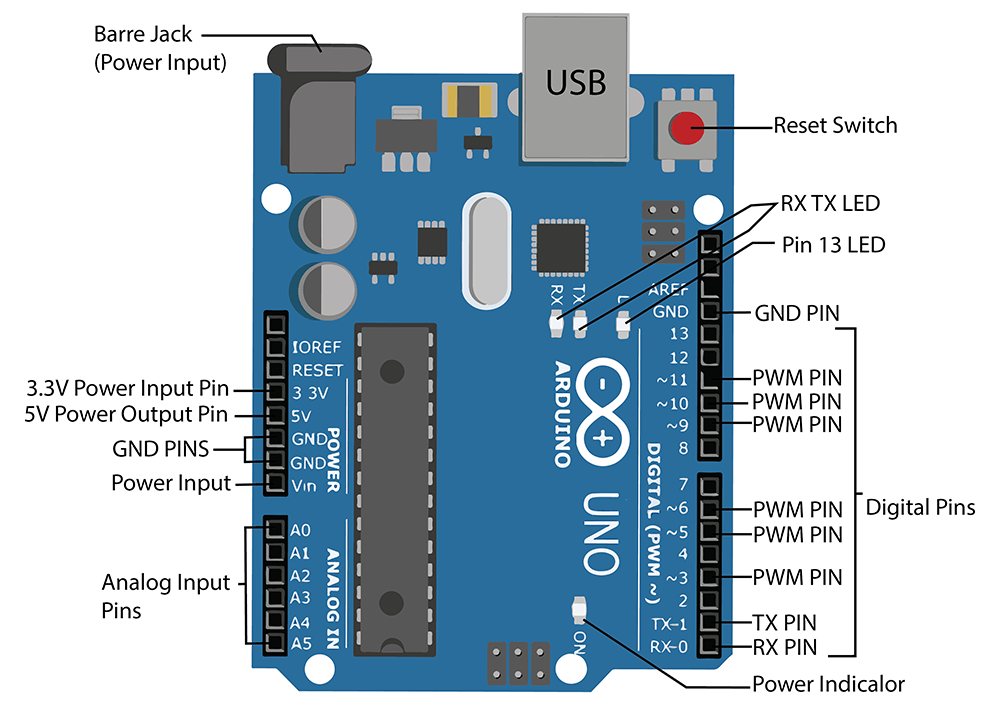


Figure 3. 1: Arduino Uno Pin Diagram

**Power:** A method for connecting to a power supply is required for every Arduino board. There are three ways to power the Arduino UNO and these are: Barrel Jack VIN Pin and USB cable. Use a power supply no higher than 12 Volts to avoid overpowering your Arduino and ultimately destroying it. For most Arduino models, a voltage between 6 and 12 volts is advised.

**PWM Pins:** These pins perform as regular digital pins but can also be utilized for a process known as pulse-width modulation (PWM). They are used as analog output (like fading an LED in and out). PWM pins are of an Arduino are 3, 5, 6, 9, 10, & 11, and gives an output of an 8-bit PWM with the function analog Write ().

**RX – TX**: These pins are used for serial communication and can connect to both computers and other Arduino boards.

**Reset Button:** The code loaded on the Arduino can be restarted using this button.

**Power Indicator LED:** Every time you plug your Arduino into a power source, this LED ought to turn on. There is a good possibility that something is amiss if this light does not turn on.

**Pin 13 LED:** An LED included into the Arduino Uno is wired to digital pin 13. LED turns on whenever the pin is HIGH and turns off whenever it is LOW.

**RX – TX LEDs:** When our Arduino is sending or receiving data on the RX TX Pins, these LEDs will provide us with some excellent visual cues.

**Memory:** The memory of this Atmega328 Arduino microcontroller includes flash memory-32 KB for storing code, SRAM-2 KB EEPROM-1 KB.

**Install the Arduino Desktop IDE and Setup**

As we have earlier mentioned, apart from the Arduino board, to start working with the microcontroller, you will need to install the IDE. Depending on your operating system, choose one of the following links to get step-by-step instructions.

Step 1: Navigate to the following URL <https://www.arduino.cc/en/software>

Select the operating system of choice. For the purpose of this module, we are going to select Windows.

Step 2: Navigate to the location where the downloaded executable file is and double click it to start the installation process. Accept the licence agreement by clicking on “I Agree”.

Step 3: Select the user from the screen shown on Figure 3.2.

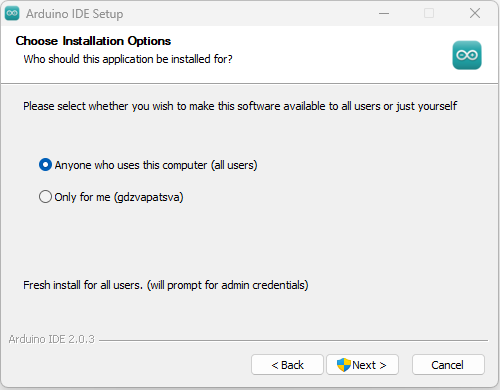


Figure 3. 2: Installation options

Step 4: Select the folder in which you want to install the IDE. The default one is C:\Program Files\Arduino IDE and click next.

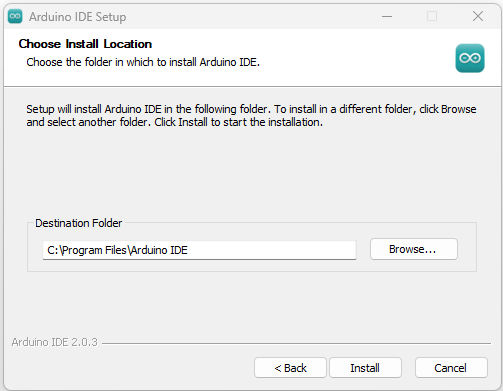


Figure 3. 3: Installation path

Step 5: Installation process goes on and when done click finish on the window as shown on Figure 3.4.

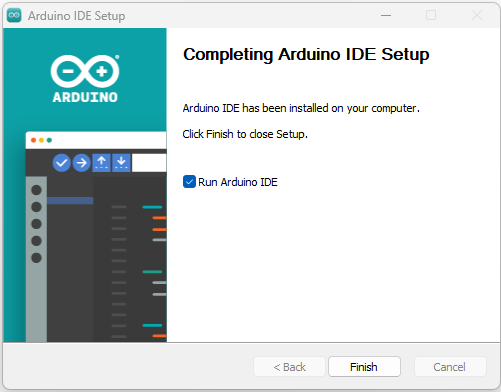


Figure 3. 4:Arduino IDE Setup completion

Immediately, you will see the Arduino IDE as shown in Figure 3.5.

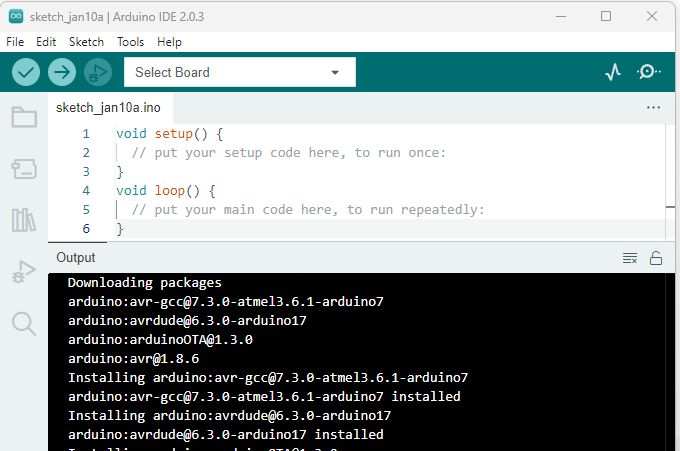
****

Figure 3. 5: Arduino IDE

### 3.1.2 Expand the term GPIO

The term "GPIO" stands for General-Purpose Input/Output. A key feature of the Arduino board is its row of GPIO pins along the top and bottom edges. An Arduino board is physically connected to the outside world via these pins. The simplest way to think about them is as switches, which you can turn on or off (input) or the Arduino can turn on or off (output).

### 3.1.3 Compare and contrast some of the major advantages of python compared to other programming languages

In this section we are going to compare C++ and Python. In level 2, you learnt Python programming and it is easy for us to refer as we carry out or comparison. Both C++ and Python are popular and widely used **high level** programming languages that offer a wide variety of programming capabilities in addition to being **versatile** and **object-oriented**. Both C++ and Python allows single and multiple inheritance.

In terms of syntax, usability, simplicity, and overall programming philosophy, the two programming languages are significantly dissimilar. We are going to tabulate the differences

|  |  |  |
| --- | --- | --- |
| **Parameter** | **C++** | **Python** |
| **Extension** | C++ program are saved with .cpp extension. | Python programs are saved with .py extension. |
| **Code** | Tends to have long lines of code. | Python has fewer lines of code. |
| **Structure** | Use curly brackets | Use indentation |
| **Compilation** | C++ is precompiled. | Python is interpreted. |
| **Speed** | C++ is faster once compiled as compared to python. | Python is slower since it uses interpreter |
| **Nature of variables** | C++ is statically typed. You can not declare a variable without a data type. | Python is dynamically typed. You do not need to indicate data type on declaration |
| **Variable scope** | In C++, the scope of variables is limited within the loops. | Accessible even outside the loop. |
| **Memory Management** | C++ does not support automatic memory management (no garbage collector but can be implemented manually) | Python offers automatic memory management (garbage collector) |
| **Functions** | C++ accepts and returns a predefined type of value according to the definition. | There is no limitation on the type of the argument and the type of its return value. |
| **Common Usage** | Emmbedded systems, compilers, Databases eg MySQL, Operating Systems eg Window, Linux, Search Engines, Banking Apps etc | Data Analysis and Machine learning, Web development, Software testing |

The choice of the language depends on the programmer and level of appreciation for programming. For beginners, Python is usually the most appropriate. However, if you are more worried about speed, C++ is the best. In order to compensate for Python's fewer desirable features, developers often combine C++ modules with Python. Additionally, calling C++ from Python leads to low-level capabilities.

### 3.1.4 Explain and identify where the GPIO pins are located

We must configure the switch as input if we wish to read the state, sensor data, etc., from it. And we must set it up as an output if we want to control the LED brightness, motor rotation, display of text, etc. The 14 digital IO pins (pin 0 -pin 13) of the Arduino Uno board can be utilized for input and output devices. The analog pins (pin A0 to A5) can be used as digital IO. Figure 3.6 illustrate the GPIO pins on the UNO board.

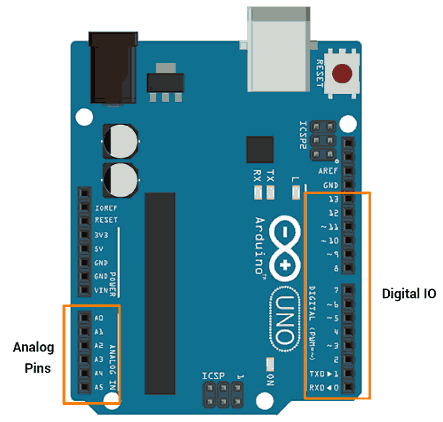


Figure 3. 6: Uno GPIO pin layout

### 3.1.5 Differentiate between a compiler and an interpreter

Compilers and interpreters convert high level languages (Source Codes) into machine code that computers can understand. Computer programs are usually written in high level languages that can be understood by humans. Compilers and interpreters are used to translate these source codes into machine language in this process. A compiler translates the entire source code in a single run whereas an interpreter translates the entire source code line by line. A detailed explanation of compilers and interpreters is given in section 4.2.1 and 4.2.4 respectively

### 3.1.6 Explain the purpose of the GPIO pins

Digital pins on an Arduino can be set up as outputs to power output devices. These pins need to be set up to be used as output. The **pinMode()** function, which determines whether a pin is an input or an output, is used to configure these pins.

* **pinMode(pin no, Mode)**
* This function is used to configure GPIO pin as input or output.
* pin nonumber of pin whose mode we want to set.
* ModeINPUT, OUTPUT or INPUT\_PULLUP

**e.g.**pinMode (13, OUTPUT)  //set pin 13 as output

Despite the Arduino pins being capable of supplying and sinking current up to 40 mA, it is not sufficient to drive motors, relays, etc.

**NOTE**

*Use resistors when connecting devices to Arduino output pins. The Arduino pin or IC will be damaged if any connected device draws more than 40 mA from the Arduino.*

Depending on the voltage level, these pins produce HIGH (5V or 3.3V) or LOW (0V) outputs. Using the **digitalWrite ()** function, we can set these pins to output.

* **digitalWrite (pin no, Output value)**

The **digitalWrite()** function is used to set output as HIGH (5 V) or LOW (0 V)

pin no number of a pin whose mode we want to set.

Output value HIGH or LOW

**E.g**., **digitalWrite (13, HIGH)**

Let’s us also discuss how we can read data from a device. Digital pins must be set up as inputs to read data from sensors or from any other type of device or circuit. Arduino pins are set to be digital inputs by default. Therefore, there is no need to set pin as input. Alternatively, the **pinMode()** function is utilized to set a pin as a digital input. The **digitalRead()** function can be used to read data from GPIO pins.

DEMONSTRATION: USING PICTOBLOX (SCRATCH) TO PROGRAM THE ARDUINO

TO LIGHT UP AN LED

void setup()

{

pinMode(13, OUTPUT); // sets the digital pin 13 as output

}

void loop()

{

digitalWrite(13, HIGH); // sets the digital pin 13 on

delay(1000); // waits for a second

digitalWrite(13, LOW); // sets the digital pin 13 off

delay(1000); // waits for a second

}

### 3.1.7 Discuss the major characteristics of the python programming language as an interpreted one

### 3.1.8 Define the term physical computing

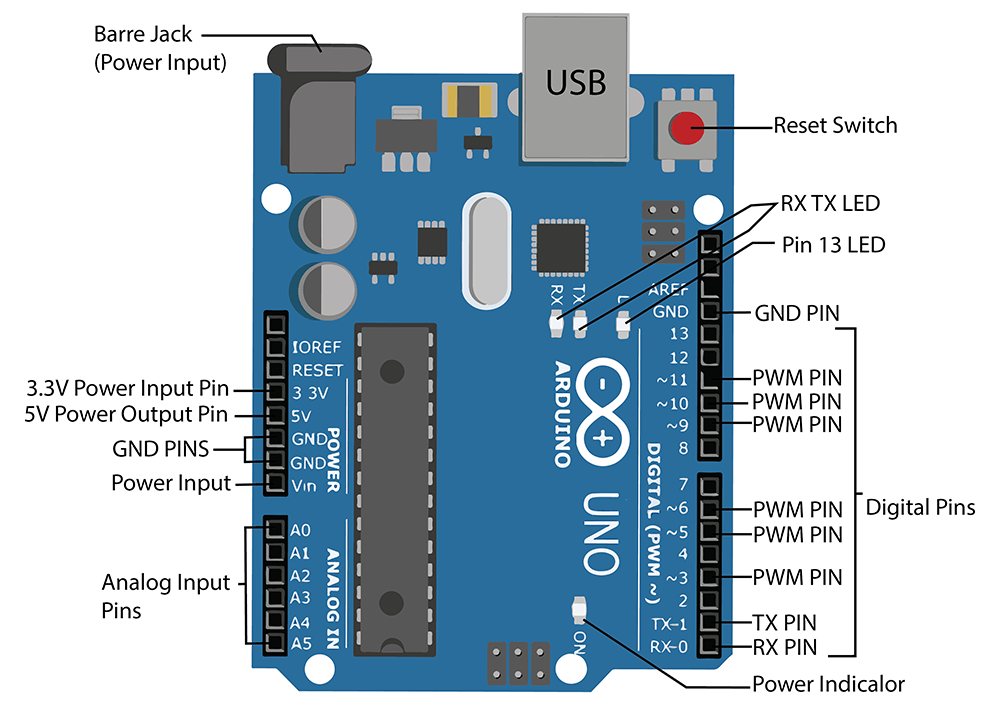
### 3.1.9 Differentiate between a shell and an IDE

### 3.1.10 Read and interpret a Pi GPIO Pin guide

### 3.1.11 Define the term physical computing

### 3.1.12 Explain what the terms 3V3, 5V, GND GP2 means on the GPIO board

### 3.1.13 Discuss how Arduino is used to enable physical computing



# Module 4 Programming tools and utilities

After you have completed this module, you should be able to :

* Define *the* term compiler
* Define the term source code
* Explain what a decompiler is used for
* Define the term interpreter
* Explain the difference between a compiler and an interpreter
* Explain what a binary is and when it is produced
* Explain the difference been C and C++ compiler
* List and Explain the basic three stage compiler design (Front Middle Back end)
* Install and configure C/C++ compiler on desktop PC
* Define the term IDE
* Explain what an IDE is used for
* Install and configure IDE on desktop PC
* Explain what the acronym GCC stands for
* Explain what the GCC collection contains
* List compilers included in the GCC
* Explain the difference between GCC and MinGW
* Install and configure GCC C/C++ compiler on Raspberry Pi
* Install and configure Geany on Raspberry Pi
* Define the term debugging
* Explain why an application needs to be debugged
* List common debugging techniques (Interactive, Print, Remote)
* Define the term breakpoint as it relates to debugging
* Define the term stepping as it relates to debugging
* Explain what the GDB tool is used for
* Install and configure MinGW on desktop PC
* Locate the GDB application in the MINGW installation
* Add the MINGW bin directory to the operating system path environment variable
* Compile source code wit the -g flag in IDE
* Explain the concept of a debugger frontend
* Install and configure gdbgiu on Desktop PC
* Start gdbgui application
* Load a compiled C/C++ binary (.exe)
* Run the loaded binary
* Step through the running binary line by line
* Investigate (watch) the values of local variables
* Explain what Git is and what Git is used for
* Discuss the three main goals of the Git VCS
* Explain the relationship between local Git directory and directory located on server
* Discuss the software license used for Git
* Name and explain the advantages and disadvantages to the software license used by Git
* Explain why Git was created
* Explain the relationship between Git and hosting providers like GitHub
* List major open-source projects using Git
* Install and configure Git on desktop PC
* Install and configure Git on Raspberry Pi
* List and explain common terminology associated with distributed VCS’s

**4.1**

**Introduction**

In chapter 1.2.1, we discussed about the different types of language translators. Translators convert programs written in high-level languages into machine code that a computer understands. To bring you up to speed, a language translator in software programming terms, is a generic term that could refer to a compiler, assembler, or interpreter. In this chapter, we are going to explain more about the C or C++ compiler.

### 4.1.1 Revisiting version control

You will recall that we touched on version control in level 2. We mentioned that version control is used to track and control changes to source code. Version control systems are software tools that help software teams manage changes to source code over time. As part of this course, you will be required to develop applications or programs as individuals or in teams. If there is no system to track the changes done regularly, you might end up submitting a version which was developed earlier after several changes. In essence, keeping track of changes is difficult. We do know that you can do that manually by changing file names but still requires constant tracking of the names. Also rolling back will be a challenge. Version control comes in hand to deal with these issues. In this chapter, we will do practical on using git technology as our version control software.

**Importance of version control systems**

* **Easy Modification of the codebase** -version control system makes it easier for continuous process of modifying programs.
* **Reverting Errors**- Emergency hot fixes, normal maintenance, upgrades, and new features with possibly conflicting development schedules can all benefit greatly from version control.
* **Collaboration**- Through version control, developers and programmers can collaborate easily. Version control enables team members to work independently and concurrently on the same document without interfering with each other's contributions.
* **Backup**- A distributed VCS acts as a backup.
* **Traceability**- A system called traceability shows all the revisions and modifications that have been performed throughout time.
* **Reduction Of Duplication and Errors-**Version control can reduce the duplication of multiple and/ or out-dated versions of any given document. As a result, it will lessen errors brought on by information that is shown inconsistently across many publications.

**4.2 Building and running C/C++ applications**

### 4.2.1 Define *the* term compiler

**VOCABULARY**

A compiler is a language processor that reads a whole source program written in high-level language in one go and converts it into an equivalent program written in machine code.

Because they must instantly transform higher-level code into lower-level machine language and save the executable object code to memory, compilers can be slow. A compiler generates machine code for a particular Instruction Set Architecture (ISA), which is unique to each processor. For instance, without a specific compiler, it is impossible to compile code for the x86 architecture and run it on the Million instructions per second (MIPS) architecture.

If the source code in a compiler is error-free, it will translate correctly to object code. If there are any mistakes in the source code, the compiler specifies the errors with line numbers at the end of the compilation. Before the compiler can correctly recompile the source code, the errors must be fixed. Examples of languages which uses compilers are C, C++, C#, Java,  Erlang, Haskell, Rust, and Go.

**Advantages of compilers**

1. **Independence**-No other program or application is required to run the executable file of source codes.
2. **Optimisation**-The compiled program is well optimized and runs faster.
3. **Data Security**-The compiler generates executable files that can be executed on any other system.
4. **Speed-**Compiler are faster than interpreter

**Disadvantages of Compilers**

1. **Hardware Specific**
2. **Time consuming-** Compilation process takes time before an executable file is produces
3. **Extra memory**-Because source code object code file and executable files
4. **Debugging Difficulty**- All errors are shown at once.

### Define the term source code

In general, programming statements written by a programmer using a text editor or visual programming tool and then saved in a file are referred to as source code. When the source code is compiled using a C compiler, the output, a compiled file, is referred to as object code.

It's common to refer to a computer program's source code and object code as its "before" and "after" versions respectively. Since there is only one form of the code, the words source code and object code do not apply to script (noncompiled or interpreted) programming languages like JavaScript. Figure 4.1 illustrates the conversion from source code to object code.

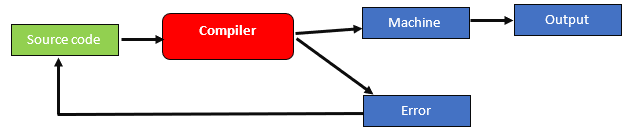


Figure 4. 1:Source code to object code

To write a source code, all you really need is a simple text editor like Notepad++, notepad or any other. This way, source code can be saved as plain text with the correct file name ending for the programming language. So, if you find a file with the ending “.cpp” on your hard drive, then it’s a text file, containing code in the C++ programming language.

Here is an example of C++ language source code:

**EXAMPLE 4.1**

#include <iostream>  
using namespace std;  
  
int main() {  
 cout << "Hello, World! \n" ;  
 cout << "This is a test program!" ;  
 return 0;  
}

It is obvious to everyone who isn't a computer programmer that the code on Example 4.1 has anything to do with the text being printed. "Hello World” and “This is a test program!" . When executed the output is as follows:

C:\ CLionProjects\untitled\cmake-build-debug\test.exe

Hello, World!

This is a test program!

Process finished with exit code 0

We are not going to explain the code for now, but we wanted to show what will be printed.

The object code file contains a sequence of machine-readable instructions that is processed by the CPU in a computer. Operating system or application software is usually in the form of compiled object code.

Here is a list of some popular C++ compilers/IDE

* [C++ Builder](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#1_C_Builder)
* [Microsoft Visual C++](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#2_Microsoft_Visual_C)
* [Eclipse IDE](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#3_Eclipse_IDE)
* [Codeblocks](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#4_Codeblocks)
* [Dev-C++](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#5_Dev-C)
* [NetBeans IDE](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#6_NetBeans_IDE)
* [Cygwin](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#7_Cygwin)
* [GCC](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#8_GCC)
* [MinGW](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#11_MinGW)
* [CodeLite](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#12_CodeLite)
* [Clang C++](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#14_Clang_C)
* [Clion](https://www.softwaretestinghelp.com/best-cpp-compiler-ide/#15_Clion)

**C++ Output**

In C++, the cout object belongs to the iostream object type. In the iostream header file, it is declared. It is used to display the output to the monitor, which is the typical output device. Using the insertion operator (), the information required to be displayed on the screen is added to the standard output stream (cout) using the insertion operator(<<). The **"c"** in cout refers to "character" and "out" means "output".

cout means "character output". The << operator can be used more than once with a combination of variables, strings, and manipulators (like endl). We are not going to go deeper into this for now. Example 4.1 demonstrate how to display text in quotes (string) using cout.

**Outputting numbers in C++**

To print the numbers, we use the same cout object but without using quotation marks. Look at the example 4.2

**EXAMPLE 4.2**

#include <iostream>

using namespace std;

int main() {

int number\_1=22;

int number\_2=19;

cout<<number\_1;

cout<<number\_2;

return 0;

}

OUTPUT

Graphical user interface, application

Description automatically generated

Looking at our program, you will notice that the numbers 22 and 19 are not in quotation marks, so when we want to print them, we make use of the names of spaces in which these two values are kept which in our case is number\_1 and number\_2 for values 22 and 19 respectively.

Our only challenge is now that the two numbers are printed joined together (concatenated) We can rewrite the output statements as follows:

cout<<number\_1<< endl;

cout<<number\_2<< endl;

A new line is added using the endl manipulator. For this reason, each output is shown in a separate line.

The << operator can be used more than once if we want to print different variables, strings and so on in a single statement. For example:

cout<<"The first number is : " << number\_1<< endl;

cout<<"The second number is : "<<number\_2<< endl;

**Output**

The first number is : 22

The second number is : 19

Inputting data to the C++ program

When using a conventional input device, like the keyboard, C++'s cin function accepts structured input. The >> operator and the cin object are used to take input. Let us look at example 4.3 which asks the user to input a number.

**EXAMPLE 4.3**

#include <iostream>

using namespace std;

//main function

int main() {

int num;

//displaying output

cout << "Enter an integer: ";

//accepting input

cin >> num; // Taking input

//displaying out

cout << "The number is: " << num;

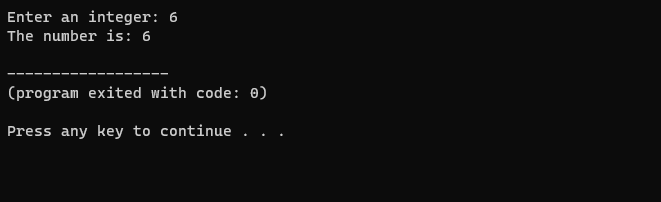
return 0;

}

Output



Type in any number for instance 6. The final output will look as follows:



In the program, we used cin >> num; to take input from the user. The input is stored in the memory location called num. We use the >> operator with cin to take input.

**Accepting multiple inputs**

Lets assume in our previous code we wanted to accept two numbers. What it means is these numbers must be stored in two memory locations and when we accept each number will be stored in its own memory location as follows:

**EXAMPLE 4.4**

#include <iostream>

using namespace std;

//main function

int main() {

int num, num2;

//displaying output

cout << "Enter an integer: ";

//accepting input

cin >> num >>num2; // Taking input

//displaying out

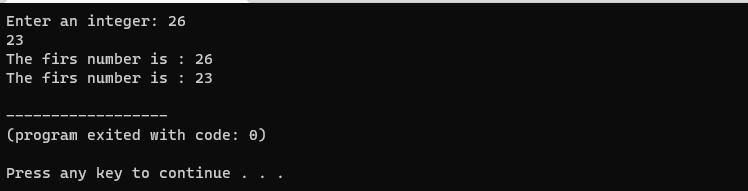
cout << "The firs number is : " << num << endl;

cout << "The firs number is : " <<num2;

return 0;

}

Output



### 4.2.3 Explain what a decompiler is used for

In software programming, a decompiler converts executable programs or machine language into a form programmers can understand. It performs the operations of a compiler, which translates source code into an executable format, but in reverse. A decompiler’s recipient is a human user, whereas the compiler’s is the machine.  Decompilation is a type of reverse-engineering that performs the opposite operations of a compiler.

**NOTE**

To convert a programming source code into a working program, it must first be compiled -- i.e., converted into a series of binary bits or digits (that is, 1s and 0s) that can be understood by the computer. This operation can be reversed by decompiling the final program (which is why decompiling is described as a type of **reverse-engineering)**.

Decompilation can be used unethically to copy source code for reuse or adaption without the owner's consent. A decompiler can be useful in some cases for the following purposes:

* Recovery of lost source code to archive or maintain the code
* Debugging programs
* Antivirus capability to find vulnerabilities in the program

Some common examples of decompilers includes:

* IDA Pro
* Hex-Rays Decompiler
* CFF Explorer
* Hiew

### 4.2.4 Define the term interpreter

**VOCABULARY**

An interpreter program executes other programs directly, running through program code and executing it line-by-line.

An interpreter is a program that also translates a high-level language into a low-level language, but it does so while the program is running. You write the program in a text editor or something similar, then tell the interpreter to run it. It runs the program one line at a time, translating each line as it goes: it translates the first line and runs it, then the second line and runs it, and so on. Python and BASIC are good examples of an interpreted language.

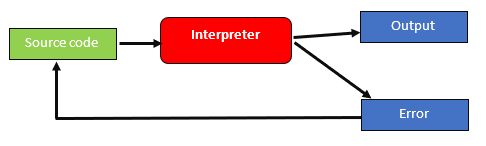


Figure 4. 2:How interpreters work

**Advantages Of Interpreter**

1. **Cross platform**- We can directly share the source code in interpreted languages, which can run on any system without any system incompatibility issues.
2. **Easier to debug**- Interpreters make code debugging easier because they read the code line by line and return the error message immediately.
3. **Less Memory** and steps- Interpreters change code into output files.
4. **Execution control**- Because an interpreter reads code line by line, you can halt execution and edit the code at any point, which a compiled language does not allow.

**Disadvantages Of Interpreter**

1. **Slower**
2. **Dependencies file required-**The interpreter is often slower than the compiler because it reads, analyses, and converts the code line by line.
3. **Less Secure-** Interpreters do not generate executable files, so to share the program with others, we must share our source code, which is insecure and private.

### Difference between a compiler and an interpreter

Table 4. 1: Comparison of a compiler vs an interpreter

| **Compiler** | **Interpreter** |
| --- | --- |
| A compiler translates the entire source code in a single run. | An interpreter translates the entire source code line by line. |
| Compilers are faster than an interpreter. | Interpreters are slower than the compiler. |
| CPU utilization is more. | CPU utilization is less as compared to the compiler. |
| Both syntactic and semantic errors can be checked simultaneously. | Only syntactic errors are checked. |
| The compiler is larger. | Interpreters are often smaller than compilers. |
| The localization of errors is difficult. | The localization of error is easier than the compiler. |
| The compiler is used by the language such as C, C++. | An interpreter is used by languages such as Java and Python. |

### 4.2.6 Explain what a binary is and when it is produced

A binary file is one that contains data in a format known as binary, which consists of a series of bytes that are sequential and each eight bits long. The material needs to be interpreted by a software program or hardware processor that is aware of how it is formatted and how to read the data in advance. Among the many different file kinds that can be stored as binary files are executables, libraries, images, databases, archives, and many more.

Binary files often include a header indicating the type of file. Application developers and other software developers commonly use binary files. An executable program is generally identified as a binary file with an extension such as .bin or .exe. Binary files are often described as executables or as compiled applications. Files of this type are also known as object code.

### 4.2.7 Key differences between C and C++ compiler

Applications, games, databases, operating systems, and games are commonly developed in C and C++. Although C and C++ sound similar, their features and usage differ. The C programming language supports objects and classes. C++, on the other hand, is an enhanced version of C that supports object-oriented programming. C is a procedural-oriented language and C++ is an object-oriented language, which is the most important difference between them. Table 4.2 demonstrates the key differences between C and C++ programming languages.

Table 4. 2: Differences between C and C++ programming

|  |  |  |
| --- | --- | --- |
| **Key** | **C** | **C++** |
| Developer | C developed by Dennis Richie between 1969-1973 at AT&T Bell Labs. | C++ developed by Bjarne Stroustrup in 1979. |
| Object Oriented Programming | C does not support OOPS concepts like polymorphism, encapsulation, and inheritance. C is procedural. | C++ is a object oriented programming and supports OOPS concepts like polymorphism, encapsulation, and inheritance. C++ supports both procedural and OOP. |
| Function | C is function driven language. | C++ is Object driven language. |
| File extension | The file extension of a program in C language is .c. | The file extension of a C++ program is .cpp. |
| Header | C uses <stdio.h> header file for input/output operations. | uses <iostream> header file for input/output operations. |

### 4.2.8 Basic three stage compiler design (Front Middle Back end)

A compiler typically has six states, and the high-level language source code that it receives is processed by each state individually to produce machine-understandable code or object code as an output. The three main stages of the compiler are backend, middle end and front end.

**Front End Phase**- Phases or parts of phases in the front end depend on the source language and are independent of the target machine. The main activity involves analysis of source code into intermediate code. These generally consist of lexical analysis, semantic analysis, syntactic analysis, symbol table creation, and intermediate code generation.

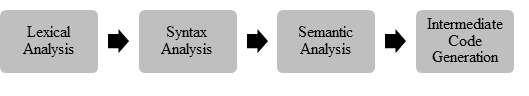


Figure 4. 3: Front End Phase

**Lexical Analysis** -Convert a program into sequence of tokens.

**Syntax Analysis** - Recover the structure portrayed by utilizing arrangement of tokens from past scanner.

**Semantic Analysis-**Ensures program has a well-defined meaning.

**IR Generation-**Compilers create an explicit low-level or machine-like intermediate representation that should be straightforward to deliver and straightforward for the target machine to understand.

**Code Generation-**The compiler converts the optimized intermediate code to the machine code dedicated to the target machine.

**Middle End**

Independent of the desired CPU architecture, the middle end optimizes the intermediate representation (IR). A source code/machine code independence is intended to make it possible to share generic optimizations between versions of the compiler that support different languages and target processors. Middle end performs target-independent optimizations on the IR (e.g., removal of useless or unreachable code, constant propagation, hoisting code from out of a loop, loop unrolling, etc.)

**Back End**

The back end of compilers includes the parts that are independent of the source language and dependent only on the target computer. In the back end, code generation and essential aspects of code optimization phases, along with error handling and symbol table activities are also provided.

Figure 4. 4: Back End compiler phase

In summary, we have learnt about compilers and their duty being to change source code in low level machine code at once. Compliers consists of front, middle and back end. The middle takes the output of the front end, transforms it somehow, then feeds that to the back-end. The [front end converts the source code to an intermediate representation](https://opensource.com/article/18/4/introduction-python-bytecode) which is not language specific. The backend converts its input to machine instructions for the target actual or virtual machine.

# FORMATIVE ASSESSMENT 4.1 INDIVIDUAL TASK

4.1.1 Define the term compiler. (2)

4.1.2 List THREE examples of compiled programming languages. (3)

4.1.3 Explain what back-end of a compiler entails. (3)

4.1.4 List 3 advantages and 3 disadvantages of compilers. (6)

4.1.5 Define the term interpreter. (2)

4.1.6 With the aid of diagrams, differentiate a compiler and interpreter. (12)

**Total :28 Marks**

## 4.2 C/C++ on the Desktop

**Introduction**

In section 4.2.2, we listed some of the C++ compilers and some come bundled with the IDE. For windows platforms, the common two options are CygWin and MingW.

The [CygWin](http://www.cygwin.com/) package provides a Portable Operating System Interface (Posix) layer in Windows, allowing much Unix software to be compiled and run without modification. When you install CygWin, you get a good command shell (bash), which may be useful. The g++ compiler can also be selected from the CygWin setup utility.

A disadvantage of CygWin is that the compiler produces code that relies on the Posix layer, which in turn is provided by a CygWin DLL. Binary executables produced by the CygWin port of g++ can only be run on other machines that have a copy of that Dynamic-link library (DLL).

MinGW, a “spin-off” project from CygWin, provides an alternate port of the g++ compiler that works without a special DLL. You may have used this already, as this is the g++ typically shipped with Code::Blocks and other basic Windows Integrated Development Environment (IDEs).

### 4.2.1 Install and configure C/C++ compiler on desktop PC

1. To install C++ compiler, search for Code::Blocks in your browser.
2. On the downloads page, select download the binary release
3. Move to the Microsoft downloads section. In our case we will select the Code::Blocks IDE and the compiler. At the time of writing this book, the latest version was codeblocks-20.03mingw-setup.exe. MingW is our C++ compiler.
4. Open and run the downloaded file and it will install the IDE plus the compiler
5. You will see the following screen.

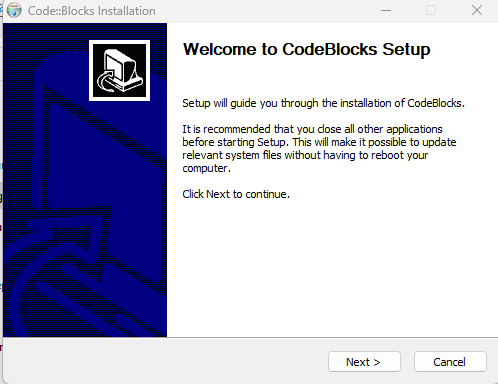


Figure 4. 5: CodeBlocks installation screen

Agree to the licence. It will also indicate the default compiler as GNU GCC compiler.

1. You will need to be very carefully with the path of MingW otherwise in some instances you run the program and it doesn’t pick up the compiler.
2. If this happens open Code::Blocks, click on compiler and select the Toolchain executables. Identify the path of your MingW and you will be good to go. Congratulations your C++ compiler is now installed

**Verify if compiler exists**

Go to the folder with MingW and type the following command: g++ --version

The output will be as follows:

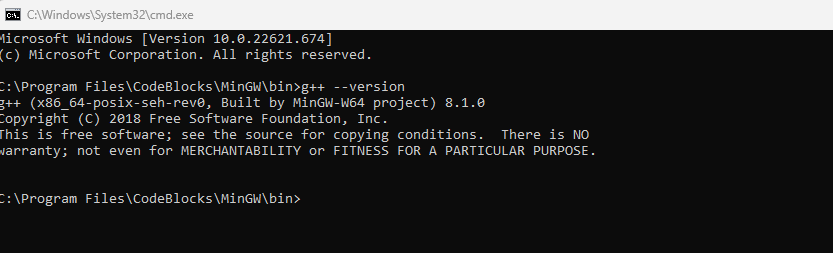


Figure 4. 6: Verifying the compiler

Type the following commands also:

* gdb --version
* make –version

### 4.2.2 Define the term IDE

**VOCABULARY**

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.

### 4.2.3 Use of IDE

A source code editor, build automation tools, and a debugger are the typical components of an IDE. Most contemporary IDEs offer intelligent code completion. Compiler, interpreter, or both may be included in some IDEs.  There are occasions when a version control system or different technologies to make creating a graphical user interface (GUI) easier are included. For usage in object-oriented software development, many contemporary IDEs additionally include a class browser, an object browser, and a class hierarchy diagram. Designed to encompass all programming tasks in one application, one of the main benefits of an IDE is that they offer a central interface with all the tools a developer needs, including:

* **Code editor:** Designed for writing and editing source code, these editors are distinguished from text editors because work to either simplify or enhance the process of writing and editing of code for developers
* **Compiler:** Compilers transform source code that is written in a human readable/writable language in a form that computers can execute.
* **Debugger:** Debuggers are used during testing and can help developers debug their application programs.
* **Build automation tools:**These can help automate developer tasks that are more common to save time.

### 4.2.4 Install and configure IDE on desktop PC

We have already demonstrated how we can install Code::Blocks which is an example of an IDE. As mentioned, another IDE which can also be use is VSCode, Eclipse, Netbeans, Geany, Komodo IDE and CLion. There are several IDEs out there and usage depends on user or organisation preferences. These IDE allow you to run languages such as C, C++, Java, Fontran and many others.

**Benefits of using IDEs**

* Faster set up
* Faster development tasks
* Continual learning
* Standardisation

We are going to demonstrate how to create a Hello World using Code::Blocks IDE.

1. To start a project, select Code::Blocks icon from the desktop or just search for Code.

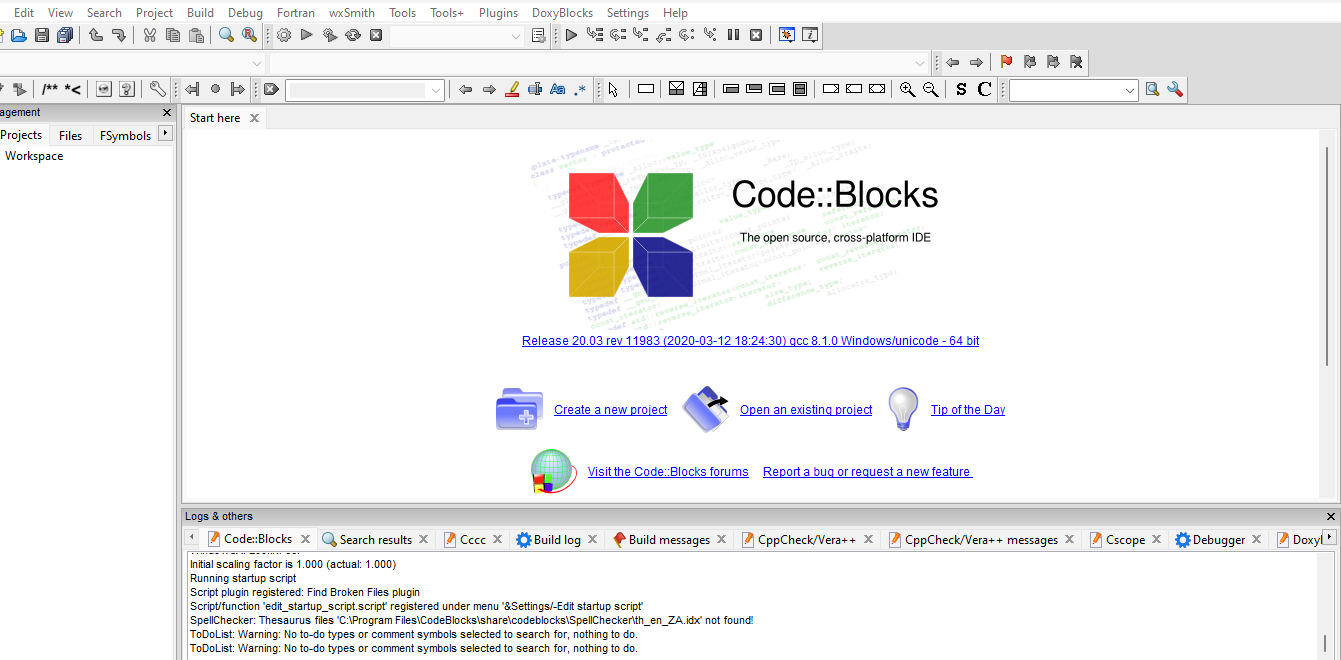


Figure 4. 7 : Code::Blocks Screen

1. Select create a new project and you will get a screen like the one on Figure 4.7 and select console application.

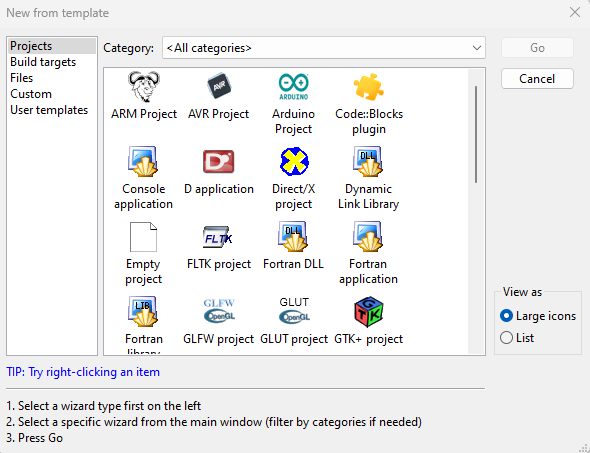


Figure 4. 8: Code::Blocks Template Window

1. Select the C++ programming language. Select the folder where the project will be saved and name of the project.
2. Navigate to the Projects workspace under the Management window and click the + sign on the Sources folder. This will open the main.cpp

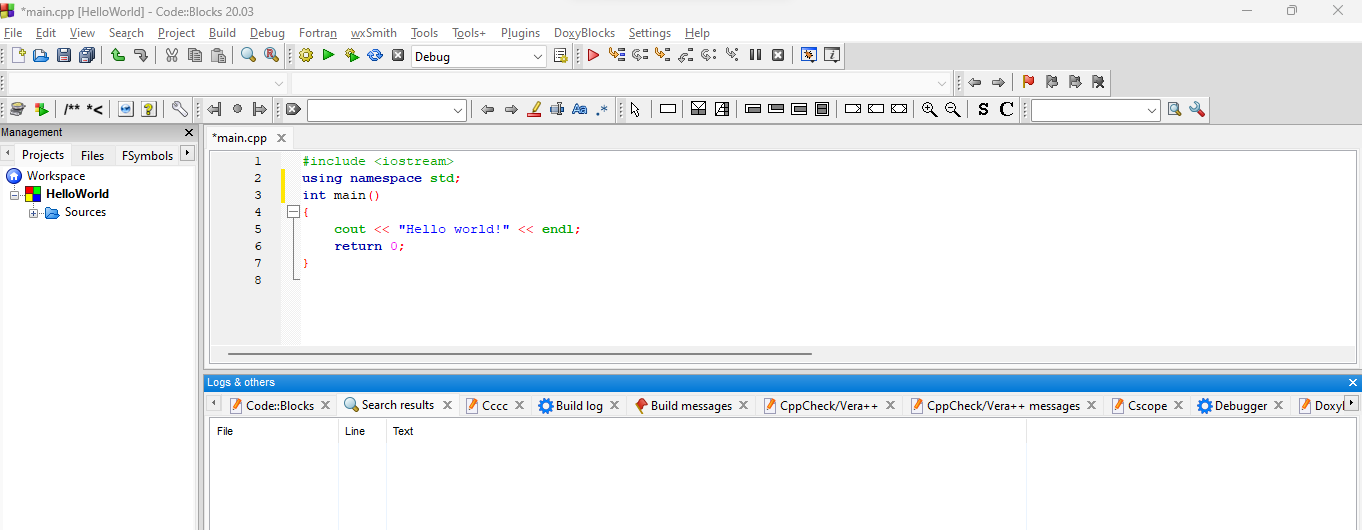


Figure 4. 9: Hello World sample project

To run the project, select build and click build and run. The sample output will be as shown in Figure 4.9.

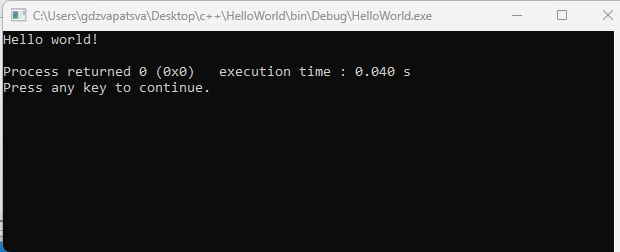


Figure 4. 10: Sample output of a C++ program

# FORMATIVE ASSESSMENT 4.2 GROUP TASK

**4..2.1** As part of software installation, you are expected to be able to install any IDE and the C++ compiler. We have listed a lot of IDE’s such as Code::Blocks, Geany

Task.

In groups, you are required to uninstall CodeBlocks and Mingw.

Reinstall Code::Blocks and Mingw

Test if gcc/g++ are available.

Run a simple hello world program.

(20)

**Total :20 marks**

## 4.3 C/C++ on the Raspberry Pi

**Introduction**

We have already introduced compiler in C++. We are now going to discuss the GCC compiler.

### 4.3.1 Explain what the acronym GCC stands for

The GNU Compiler Collection, commonly known as GCC, is a set of compilers and development tools available for Linux, Windows, various BSDs, and a wide assortment of other operating systems. GCC was created by the Free Software Foundation (FSF) and made available as totally free (as in libre) software.

### 4.3.2 GCC collection contains

GCC includes support primarily for C and C++ and includes Objective-C, Ada, Go, Fortran, and D. The Free Software Foundation (FSF) wrote GCC and released it as completely free (as in libre) software. Developers can take advantage of a wide range of capabilities provided by GCC, such as cross-platform compilation, deep code optimization, and support for different languages.

### 4.3.3 List compilers included in the GCC

gcc comes with a compiler named g++ that specifically compiles C++ programs, regardless of the file extension. Sometimes you need to use the C++ compiler even though the file extension is not a C++ extension; in this case you need to use g++.

### 4.3.4 Explain the difference between GCC and MinGW

[GCC](https://gcc.gnu.org/) (GNU Compiler Collection) is a free and open-source compiler for C and C++ (and other languages like Objective-C, Fortran, D).[MinGW-w64](http://mingw-w64.org/) is a free and open-source C library for targeting Windows 32-bit and 64-bit platforms. 'MinGW' refers to the "Minimalist GNU for Windows" project. The combination of these results in a free C/C++ compiler for Windows. Even though GCC and MinGW-w64 can be used on other platforms to generate Windows executables, the WinLibs project only focusses on building versions that run natively on Windows. The GNU Compiler Collection (GCC) port, GNU Binutils for Windows (assembler, linker, archive manager), a collection of freely distributable static import libraries and header files that enable the use of the Windows API, a Windows native build of the GNU Project's GNU Debugger, and other utilities are all included in MinGW.

### 4.3.5 Install and configure GCC C/C++ compiler on Raspberry Pi

Raspberry Pi 4 devices already have the stock GCC on it and that should include the G++. If yours doesn’t there is a command, you can invoke.

Type the following command:

sudo apt install build-essential

Alternatively, follow the steps below:

Step 1 Update raspberry Pi by typing the following command

sudo apt update && sudo apt upgrade -y

Step 2: Make sure git is installed otherwise follow steps to install git. You can check if git is installed by typing the following command:

git –version

Step 3: In this step you need to open a terminal and download a binary Gcc.

git clone <https://bitbucket.org/sol_prog/raspberry-pi-gcc-binary.git>

Step 4: Now you need to extract the archive, then move the extracted compilers to /opt and remove the repository. Use the following commands

cd raspberry-pi-gcc-binary

tar -xjvf gcc-10.1.0-armhf-raspbian.tar.bz2

sudo mv gcc-10.1.0 /opt

cd ..

rm -rf raspberry-pi-gcc-binary

**Difference between GCC and G++**

Table 1: Differences between G++ and GCC

|  |  |  |
| --- | --- | --- |
| **Parameter** | **G++** | **GCC** |
| **Uses** | G++ is used for compiling C++ | Used for compiling C |
| **Compilation** | can compile either.cpp or.c files, but they will only be treated and released as C++ files. | can compile either.cpp or.c files, but they will be treated and released as C++ or C |
| **Command for Compilation** | g++ fileName.cpp -o binary. | gcc fileName.c -o binary. |
| **File Linking in Library** | When we use the G++ command to link the object files, the files automatically link in the standard C++ libraries. | It does not happen in the case of GCC. |

### 4.3.6 Install and configure Geany on Raspberry Pi

Geany is a very lightweight code editor or IDE. Due to its extensive capability and effective use of computer resources like CPU and, most importantly, RAM, Geany is ideally suited for programming on the Raspberry Pi. On Raspberry Pi 4, Geany is already installed.

**Steps to install Geany on raspberry Pi**

Step 1: Open a terminal and run the following command:

sudo apt-get install geany

**Geany Features**

* Lightweight and fast
* Split screen editing
* Broad cross platform support
* Extensibility through plugins

# FORMATIVE ASSESSMENT 4.3 INDIVIDUAL TASK

4.3.1 Explain what GCC is. (2)

4.3.2 Differentiate MingW and GCC (2)

4.3.3 Define the term IDE. (2)

4.3.4 Write the steps to install Geany on a Raspberry Pi. (4)

4.3.5 Complete the following table to differentiate G++ from GCC. (8)

|  |  |  |
| --- | --- | --- |
| **Parameter** | **G++** | **GCC** |
| **Uses** |  |  |
| **Compilation** |  |  |
| **Compilation Command** |  |  |
| **File Linking in Library** |  |  |

**Total : 18 marks**

## 4.4 Debugging C++ applications

**Introduction**

When programmers code programs, they expect the code to work according to their plans. However, source code often breaks or does not work as anticipated. When this happens, programmers need to find where the problem is so that the code gives them the output they expect. Debugging tools are used to execute the program in a controlled setting, check the code step by step, and identify and correct the problem.

### 4.4.1 Define the term debugging

**VOCABULARY**

Debugging is the process of finding and fixing errors or bugs in the source code of any software.

There are two main categories of errors that need debugging:

1. **Compile-time**: These occur due to misuse of language constructs, such as syntax errors. Normally easy to find by using compiler tools and warnings to fix reported problems.
2. **Run-time**: These are much harder to figure out, as they cause the program to generate incorrect output (or “crash”) during execution. Example is a runtime error.

Programmers will check for the following during the debugging process, which can be carried out manually or automatically using software debugging tools:

* Syntax errors- A syntax error is a bug that occurs when a computer program has an incorrectly typed statement. Here is an example of a syntax error:

Graphical user interface, application

Description automatically generated

Figure 4. 11:Syntax error

Looking at Figure 4.11, notice the squiggly red line in line 1. The C++ word which is causing the error is <*iosteam*>. The correct spelling is supposed to be <iostream>.

* Runtime errors-occurs during runtime and could be caused by inputting wrong data type. For instance, the program which is supposed to accept a integer (number) and user types in a string.
* Logic errors-occurs when program runs but gives incorrect results

Testing and debugging work together to make sure software programs function as intended. Programmers test after finishing a segment or a portion of code to find faults and problems. Once flaws are identified, programmers can start the debugging process and try to clean up any errors in the software.

### 4.4.2 Reasons for debugging

1. Any software which is developed needs to be bug-free before releasing.
2. Competitive advantage-Bug free programs will build a good reputation for the software firm
3. Saves Time**-** Performing debugging at the initial stage saves the [time of software developers](https://www.educba.com/career-as-a-software-developers/).
4. Making sure that the software programs are serving the intended purpose

### 4.4.3 Common debugging techniques

1. Print debugging (also called tracing) - the developer watches live or recorded print statements and monitors’ flow.
2. Interactive debugging- Using debugger tools, interactive debugging enables the processing of an application's code execution one step at a time and pauses it to inspect or modify application state.
3. Remote debugging enables you to debug one or more applications on a remote machine when the IDE is running only on your local machine.

**Debugging Strategies**

Programmers employ a variety of strategies to avoid errors and shorten the debugging process. The following are a few debugging strategies that programmers frequently use.

1. Backtracking - The programmer works backwards through the source code, starting at the statement where an error symptom is found.
2. Cause Elimination Method- To identify the source of a point of failure, the software engineer compiles a list of likely causes for errors and conducts tests.
3. Shotgun Debugging - a developer's educated guesses-based debugging approach that relies on trial-and-error and crossing one's fingers.
4. Program Slicing- A series of program statements are run in the program under certain circumstances by quality assurance testers.

### 4.4.4 Define the term breakpoint as it relates to debugging

A breakpoint is an intentional halt in an application's code where execution pauses for debugging. This enables the programmer to inspect the internal state of the application at that time. We are going to discuss two types of breakpoints:

**Conditional breakpoints**- Conditional breakpoints allow you to break inside a code block when a defined expression evaluates to true. The breakpoint is triggered when something in memory meets a given condition.

**Action breakpoints**- these allows us to take an action like printing to the console when a breakpoint is at a given spot.

Note: Everything that you can do in breakpoints can still be done by the code.

**How to set a conditional breakpoint in Code::Blocks**

Breakpoints are set in the normal editor window by clicking on the “gutter” on the left-hand side of the source file and a red dot will appear.

The following breakpoint commands are available under the Debug menu:

* Debug->Toggle Breakpoint: set or clear a breakpoint at the cursor
* Debug->Remove all breakpoints: remove all breakpoints from the project. A programmer can also edit the breakpoint.

A disabled breakpoint is useful to temporarily ignore a breakpoint during a debug session without removing it.

Let us do a simple practice on debugging. We are going to use the following code which adds two numbers and display the result.

**EXAMPLE 4.5**

Step 1: Type the following code and save it.

Text

Description automatically generated

Figure 4. 12:Adding breakpoint

Step 2:Right click on line 6 and click add a breakpoint. You will see a red dot like the one in Figure 4.12

Step 3: Select Debug and click Start/Continue. Immediately you will see the console appearing without any text.

NOTE: You can set up the screen display to tile vertically so that the console and the IDE are side by side as shown in Figure 4.13.

Step 4:Select Debug and click Step into. Immediately, you will see a small triangle inside the red dot.

Step 5: Select Next Line and you will see the console displaying “Enter first number”. Enter a number in the console and press enter. Return to the IDE and select Next Line. Repeat until the steps until the yellow triangle gets to line 15 and exit.

The Message window will display line numbers as the process at each point of execution and the resulting output. See Figure 4.13.

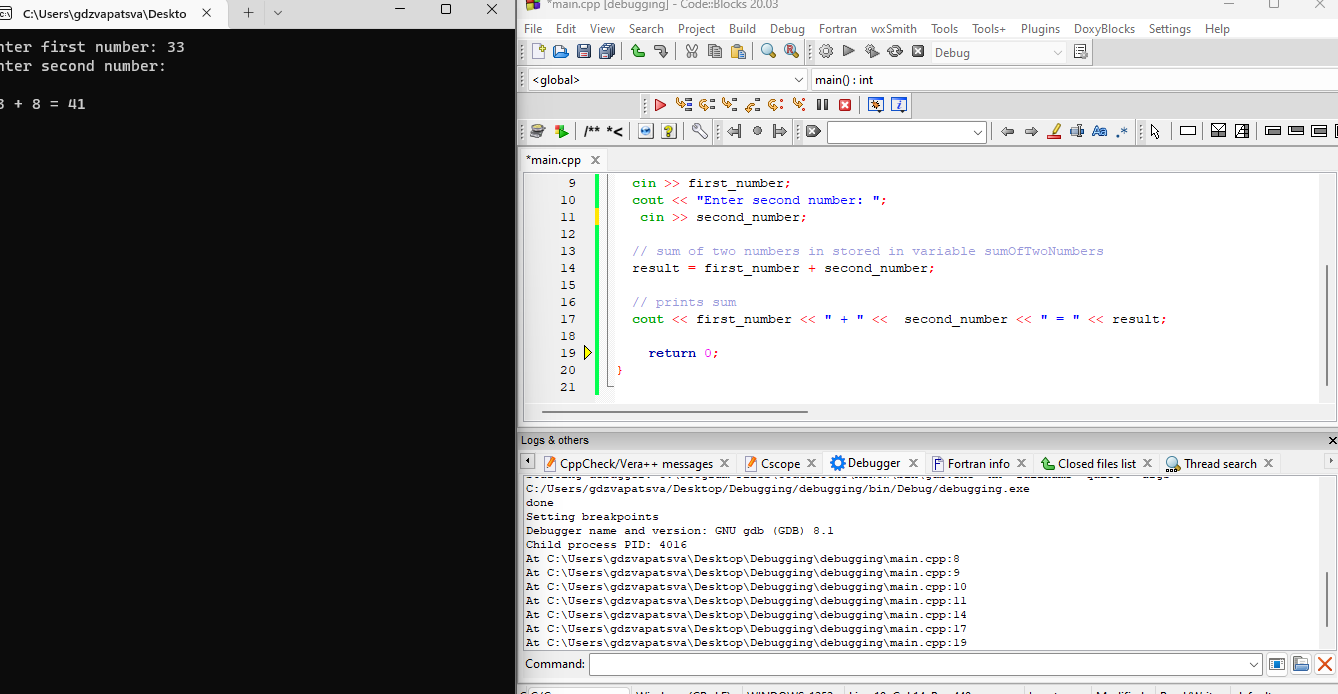


Figure 4. 13:Debugging Output

### 4.4.5 Define the term stepping as it relates to debugging

Stepping is a debugging technique that involves executing code one instruction or line at a time. Stepping will necessitate running just one more "step" of your program, where "step" can refer to either one line of source code or one machine instruction. A debugger, among other things, allows the programmer to step through code in execution, set breakpoints, view memory, change variables, and track variables. The gnu debugger is a popular serial debugger (gdb). We will discuss the GDB in section 4.4.6.

### 4.4.6 Explain what the GDB tool is used for

GDB stands for GNU Debugger and is a powerful debugging tool for languages such as C++, C and others. You can step through the code, [set breakpoints](https://www.sciencedirect.com/topics/engineering/set-breakpoints), examine and change variables, and so on. GDB is command line only, which makes it difficult to use. GDB's graphical front ends translate GUI commands into GDB text commands.

gdb allows you to poke around inside your C++ programs while they are running and also see what happens when your program crashes. GDB works with executable files, which are binary files created during the compilation process.

To check if gdb is installed, in our case we will locate it in the MingW/bin folder since it comes packed within the IDE. So, we can navigate to that folder and type the following command:

gdb –help

The output will be as follows:

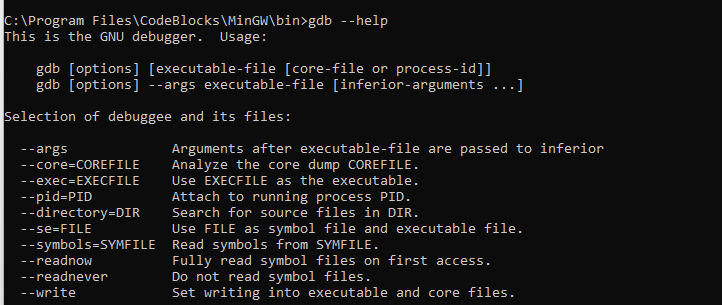


Figure 4. 14:gdb output

### 4.4.7 Install and configure MinGW on desktop PC

As you witnessed during Code::Blocks installation, you have a choice of selecting CodeBlocks with MingW installation file or without. In our case we selected one with MingW.

In case you uninstall MingW or you need to install it separate from the IDE, here are the steps:

**Step 1**: Go to the [MinGW](https://sourceforge.net/projects/mingw/) website (https://sourceforge.net/projects/mingw/) and click on Download.

**Step 2**: Double click and open the exe MinGW file and click install.

**Step 3:**When you click Continue, the MinGW installation manager will appear.

**Step 4**: Right-click on each option in the installation manager and then select Mark for installation. Click Apply changes and close. Now MingW is installed.

### 4.4.8 Locate the GDB application in the MINGW installation

To locate the gdb application from MingW, navigate to MingW\bin and you will find the application in there. Alternatively, you can type the following command in the terminal:

dir "\gdb.exe\*" /s

The output of the following code will be as shown in Figure 4.11.

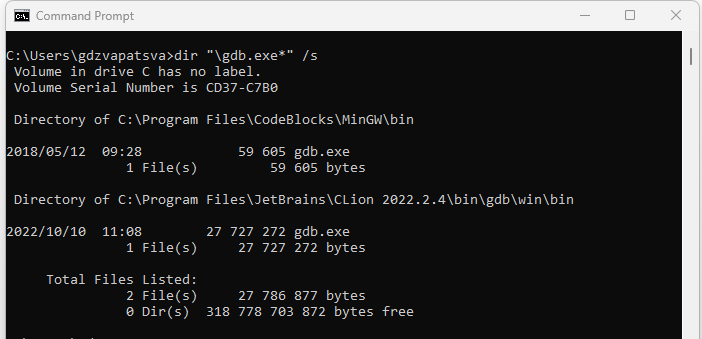


Figure 4. 15:Locating gdb application

### 4.4.9 Add the MINGW bin directory to the operating system path environment variable

Remember, in our case, we installed CodeBlocks with MingW. So, our MingW folder is located inside the CodeBlocks folder. We need to set up the path so that the compiler is available in our terminal irrespective of path. Now, if we run the following commands: gcc –version or g++ --version, we get an error message as shown in Figure 4.12

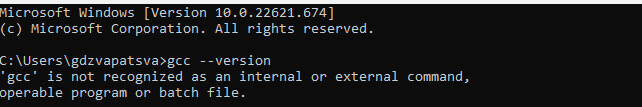


Figure 4. 16:Unrecognised command

To solve the error message in Figure 4.12, we have to set the path. So first let us navigate to CodeBlocks folder and look for MingW. Inside MingW, locate the path for bin and type cmd in the address bar. The terminal will open with the current location of bin folder. Copy the path of the terminal.

Alternatively, when in the bin folder, right click any of the files and select properties option. Copy the path under location section. Now let us set the path.

Step 1: Open environment Variables by typing env from the search bar

Step 2: Click Environment Variables Tab

Step 3: Under System variables, highlight path and click on edit

Step 4: Click New and add the copied path for bin and click OK and close any window for system variables open. Also close any command prompt window open

Step 5: Now text if the compilers are now accessible by typing any of the following command:

g++ --version

gcc –version

The terminal will give output such as one in Figure 4.13.

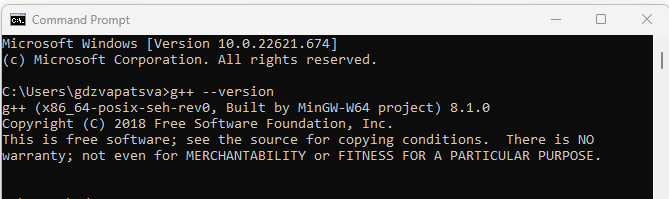


Figure 4. 17: Output of compiler on system path

Let us test our compiler

**EXAMPLE 4.6**

Open the terminal from the desktop.

Type the following command:

Notepad c\_test.cpp

You will be prompted to create a file called c\_test click yes.

Type the following c++ code

#include <iostream>

using namespace std;

int main ()

{

cout<<"1-2-3 testing";

return 0;

}

Type the following on the terminal:

g++ c\_test.cpp -o c\_test.exe

Press enter and type c\_test.exe. The output of the compiled c++ file will appear on the terminal. See sample Figure 4.14

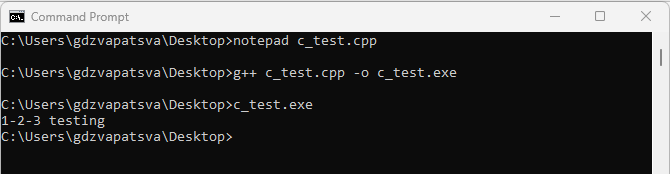


Figure 4. 18:Sample code execution and output

### 4.4.10 Compile source code wit the -g flag in IDE

To compile source code (.cpp) files using g flag is pretty straightforward as shown below.

First you will need to navigate to the folder where the project has been saved through the command prompt. For example I navigate to the path shown since I have source code in that folder.

C:\Users\User1\Desktop\ADDS>g++ -g addition.cpp -o hello.exe

Our executable file has been generated. But what was the purpose of the g flag.

The 'g' flag instructs the compiler to provide more source code information in the executable than it otherwise would. This makes using a debugger like gdb much simpler because it will be able to use variable names that are present in the source code as references. gcc -g generates debug information to be used by GDB debugger.

# FORMATIVE ASSESSMENT 4.4 INDIVIDUAL TASK

4.4.1 Define the following terms:

* 1. Syntax error
  2. Runtime error
  3. Logic errors (6)

4.4.2 Give **FOUR** reasons why debugging is important. (3)

4.4.3 Which debugging technique allows the programmer to debug one or more applications on a remote machine. (1)

4.4.4 Define the term stepping as used applied in C++ debugging process. (2)

4.4.5 Write a program which accepts first\_name and surname and then prints out the fullname. Your code should look as the one in Figure 4.19.

Text

Description automatically generated

Figure 4. 19:Sample code

Question. Add a breakpoint in line six and run the debugging process and display the output of the console as shown in Figure 4.20. Print the screenshot as displayed.

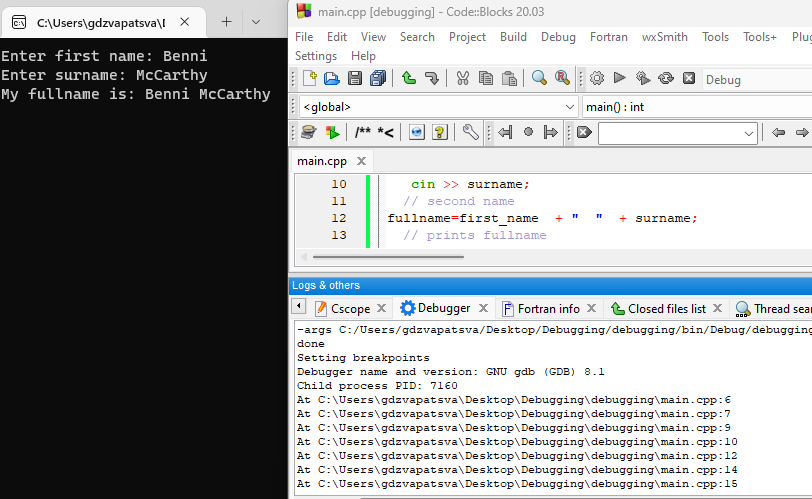


Figure 4. 20:Output

(10)

**Total: 22 marks**

## 4.5 Debugging C++ with GDBgui

**Introduction**

Developers commit a significant amount of time to debugging and maintaining existing codebases. Understanding various debugging methods is critical. Some developers still prefer more manual methods of debugging. There are also many snapshot tools that provide a detailed report of the issues and errors that occur after a specific section of code is executed. gdbgui is a browser-based frontend to gdb, the gnu debugger. You can add breakpoints, view stack traces, and more in C++ and other languages. gdbgui is another debugging tool built on top of gdb.

### Explain the concept of a debugger frontend

gdbgui provides developers with a browser-based frontend through which they can add breakpoints, view stack traces, and change the context and parameter values while the debugger is running.

### Install and configure gdbgui on Desktop PC

To install gdbgui, use pip command as follows:

pip install gdbgui==0.13.5.0

gdbgui can be problematic on Windows with incompatibility issues. The most common error you will encounter says:

RuntimeError: Windows is not supported at this time. Versions lower than 0.14.x. are Windows compatible.

The solution is to install gdbgui version less than 0.14.x

### Start gdbgui application

To start gdbgui, type the following command in the terminal:

>gdbgui

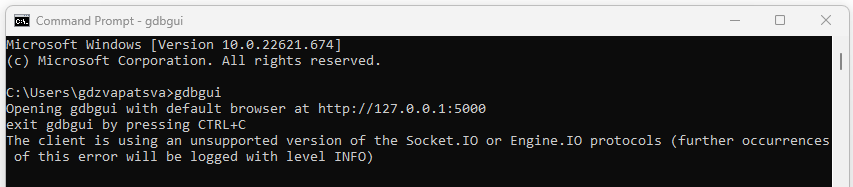


Figure 4. 21: Starting gdbgui

Once you type the command and press enter, the browser will open and

* + 1. Load a compiled C/C++ binary (.exe)

Graphical user interface, text, application

Description automatically generated

Now you can debug your executable files using a browser version.

To load the binary you just need to copy the path of the executable file and click on load binary. The full source code will appear in the browser.

### Run the loaded binary

To run the file, next to the address bar, you will find and an arrow which allows you to run the program. You will need to set breakpoint just the same way we did in Code::Blocks

### Step through the running binary line by line

While working with gdbgui, it allows programmer to type commands in the browser terminal sections.

The most frequently used GDB commands are the following:

* r: start the program
* s (step): execute next program line (after stopping); step into any function calls in the line
* n (next): execute next program line (after stopping); step over any function calls in the line
* d: delete breakpoints

### 4.5.7 Investigate (watch) the values of local variables

Use a watch condition to keep track of changes in a variable's or an expression's current value that influences a storage location's address. With one key distinction, setting watch conditions differs from setting conditional breakpoints in that they cause the program to terminate as soon as a variable's value changes from its initial value. The storage address's content is computed at the time the watch condition is set and used by the debugger to watch a variable. A breakpoint is created, the program terminates, and the content at the storage location is altered from the value it had when the watch condition was set or when the last watch condition occurred.

# FORMATIVE ASSESSMENT 4.5 GROUP TASK

4.5.1 Install gdbgui and debug the following program.

Text

Description automatically generated

(20)

**eLink**

<https://www.youtube.com/embed/Cd0BOOdGjrw>

## 4.6 The Git Version Control System

### 4.6.1 Introduction

There is often confusion between the terms *source control*, *revision control* and *version control*. The figure below provides a quick overview of source-code management. In level 2, we explained the differences by use of the following diagram shown in Figure 4:22.

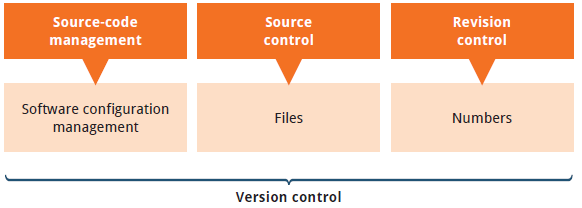
****

Figure 4. 22: Version control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.  For the examples in this book, you will use software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer.

**Types of Version Control Systems:**

* Local Version Control Systems
* Centralized Version Control Systems
* Distributed Version Control Systems

**Local Version Control Systems- It is one of the most straightforward types and includes a database that maintains revision control over all file modifications.**

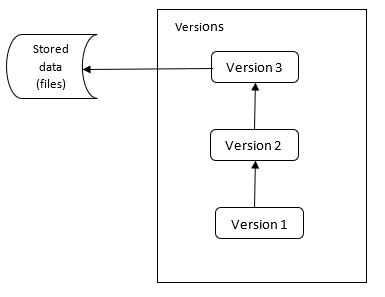


Figure 4. 23: Local Version Control Systems

**In LVCS, every patch set contains only the changes made to the file since its last version. To see what the file looked like at any given moment; it is necessary to add up all the relevant patches to the file in order until that given moment.**

**Centralized Version Control Systems- These only have a single repository globally, and each user must commit for their modifications to be reflected in the repository. With CVCS, this way, everyone usually knows what everyone else on the project is doing. Administrators have control over who can do what which might be a downfall of CVCS. Figure 4.24 shows how CVCS looks like.**

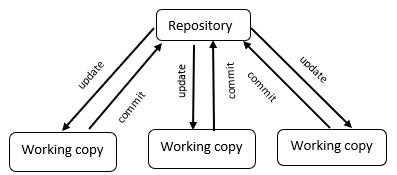


Figure 4. 24: Central Version Control System

**Distributed Version Control Systems (DVCS)-** Multiple repositories are present in distributed version control systems, and each user has a separate working copy and repository. You cannot share your modifications with others by simply committing them. This is because commit just makes those changes accessible in your local repository; to make them visible on the main repository, you must push them. Git is an example of DVCS. Linus Torvalds created Git in 2005 for the development of the Linux kernel. Git allows multiple developers to work together and supports non-linear development through its thousands of parallel branches.

Diagram

Description automatically generated

Figure 4. 25: Distributed Version Control System

### 4.6.2 Defining Git is and what it is used for

According to Git website, Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Git is easier and simpler to learn. Other similar software for version control includes Subversion, CVS, Perforce, and ClearCase. At the time of writing this book, Git was the commonly used VCS.

Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development. Developers used to submit their codes to the central server without having copies of their own. Any changes made to the source code were unknown to the other developers. There was no communication between any of the developers. However, with Git, every developer has an entire copy of the code on their local systems. Any changes made to the source code can be tracked by others. There is regular communication between the developers. For Git to flows to be completed- there must be a hosting platform for the remote repositories. Git uses GitHub.

GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Github is exclusively cloud based hosting service which enable programmers /users to manage Git repositories.

### Three main goals of the Git VCS

Git has three main goals that it attempts to achieve, and these are speed, data integrity and distributed.

**Speed**

Git handles all projects with speed irrespective of sizes. The speed might only be controlled by the network. Git servers are very efficient.

**Data integrity**

Before being stored, each item in Git is checksummed, and that checksum is subsequently used to identify the item. This means that no file or directory can have its contents changed without Git being aware of it. Git is fundamental to its ethos and has this functionality built in from the ground up.

**Distributed**

One of the nicest features of Git is that it's distributed. This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.

### 4.6.4 Relationship between local Git directory and directory located on server

Let us explain in simpler terms. When programmer(s) are working on a project, they save their file on the local machine. Once they initialise Git in that folder- the folder is known as the local repository. Any changes will need to be committed so that they are in the staging area. I hope this explanation very succinct. Here is a what you need to understand with Git. Git has three main states that your files can reside in: modified, staged, and committed:

* Modified- means that you have changed the file but have not committed it to your database yet.
* Staged- means that you have marked a modified file in its current version to go into your next commit snapshot.
* Committed -means that the data is safely stored in your local database.

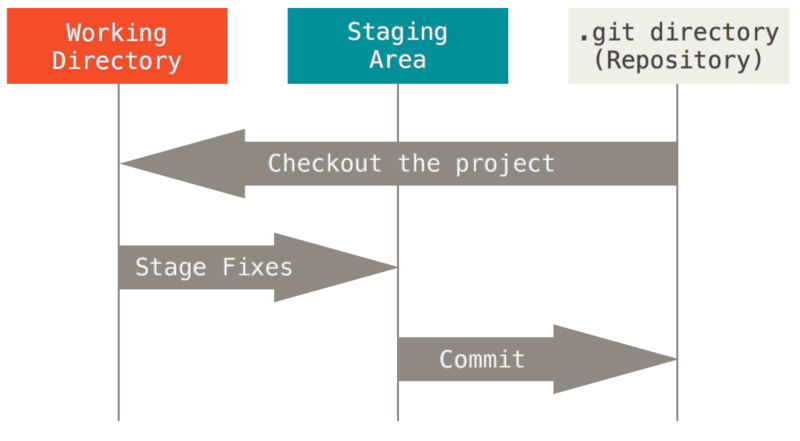


Figure 4. 26: Git workflow

(Source:https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F)

### 4.6.5 Software license used for Git

Open-source software is frequently shared via the public repositories on GitHub. You must grant people the right to freely use, modify, and distribute the software for your repository to be considered fully open source. There is no pressure on you to select a license. However, without a license, copyright rules that are in effect by default apply, which means that you maintain ownership of your source code and that no one else is permitted to copy, distribute, or create derivative works of your work. Most authors save the text of their license in a file with the name LICENSE.txt. There is a variety of licenses to choose from.

### 4.6.6 Advantages and disadvantages of software license used by Git

Git’s open-source licensing has several advantages and disadvantages.

**Advantages of Git licensing**

1. **Flexibility and agility**- Technology agility is made possible by open-source licensing, which often provides several solutions to a given issue.
2. **Speed**- The flexibility to use the community versions, get started, determine whether they can solve your business problem, and start delivering value right away is a huge benefit of open-source licensing.
3. **Cost-effectiveness**- Open-source licensing is generally much more cost-effective.
4. **Attract better talent**- Most working engineers are familiar with open source, and many of them think it will be the future of their field. Attracts large community.
5. **Collaboration culture**

**Disadvantages of Git licensing**

1. Some individuals may find it less user-friendly because we are indirectly copying the source code.
2. Support issues can also go wrong from time to time.
3. The open-source license itself is free, however there are costs related to its plugin modules and a host of other things.

### 4.6.7 Explain why Git was created

Git was created to resolve weaknesses of other version control systems for instance the centralised version control system. Git's branching features are among its greatest benefits. Git branches are inexpensive and simple to combine, in contrast to centralized version control solutions. This makes it easier to employ the feature branch workflow, which is well-liked by Git users.

### 4.6.8 Relationship between Git and hosting providers like GitHub

Most people tend to confuse Git and GitHub.The key difference between Git and GitHub is that Git is an open-source tool developers install locally to manage source code, while GitHub is an online service to which developers who use Git can connect and upload or download resources. Git is a software and GitHub is a service. To make sure that the same copies are available with the remote repository, which is found on the GitHub server, the programmer will need to push the changes. You install Git locally on your machine while GitHub is hosted on the web.

### 4.6.9 Major open-source projects using Git

Since its birth, open software has fundamentally altered the digital ecosystem. On GitHub, there are several projects that developers are working on at the same time. Here is a list of open-source projects using Git.

* Flutter
* freeCodeCamp
* First Contributors
* TensorFlow

### 4.6.10 Install and configure Git on desktop

The first important thing to do before installing git will be to check if its not installed already on your machine. Git uses commands. The command to run when checking for installed versions is git version or git - -version. These two commands produce the same result. Open command prompt and run the command.

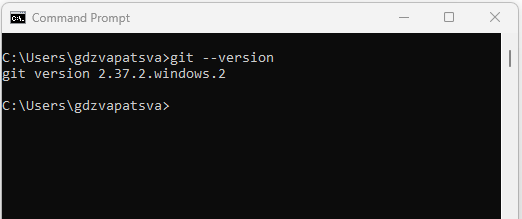


Figure 4. 27:Git checking

If Git is already installed, you will be able to see the version installed as in Figure 4.13. As you can see, my machine had Git installed and I am going to uninstall it and reinstall again for the sake of this lesson. If Git is not installed, you will get the following error message on your command prompt:

*“'git' is not recognized as an internal or external command, operable program or batch file.”*

Now you can go ahead and follow installation steps.

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website.

Steps to Install Git on Windows Desktop

1. Navigate to the following link <https://git-scm.com/download/win> and download Git
2. Open the executable and run the file. You will see license agreement screen as shown in Figure- 4.28. Click Next.

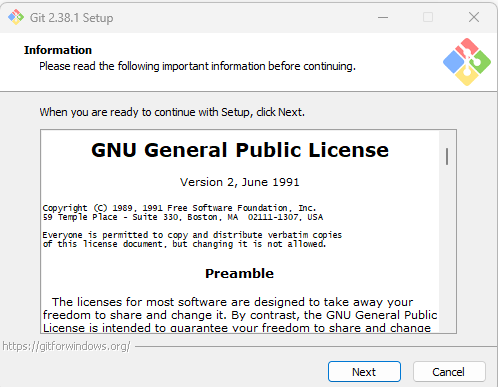


Figure 4. 28: Git License Agreement

1. Continue to click Next. As for setting the path: leave the default one.

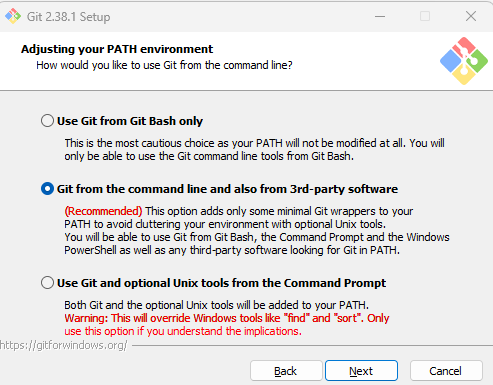


Figure 4. 29:Selecting Git Path environment

1. Continue to click Next (Leave the default options selected). Once complete run the git – version/git version. You will see the new version installed.

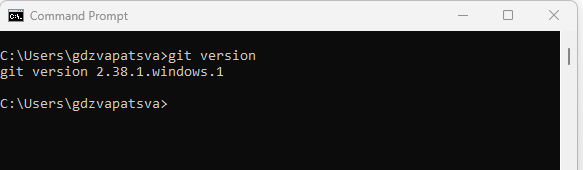


Figure 4. 30:Git Version

So, you can see that, earlier, I had git version 2.37.2 and now I have installed version 2.38.1.

Please take note, the version will change but at the time of writing this book, the version I installed was 2.38.1

**Git configuration**

1. Open command line and type the following command to configure your Git username, where <your name> will be your GitHub username.

git config –global user.name “your name”

1. Add the user email by typing:

git config –global user.email “your\_email-address”

1. To check if the configuration is successful, type the following command:

git config --list

You will see the username and email that you used to configure your Git.

1. The last important thing is to go to GitHub and sign up for an account. Remember the email address you used must be same so that your Git on your machine is linked to your GitHub account.

When you are done setting up your git, execute the following command to see that your username and email has been successfully configured:

git config --list

In my case my output shows as follows.

Text

Description automatically generated

Figure 4. 31: Git config output

The last 2 lines at the bottom of Figure 4.31 shows my git name and my git email which I used. In your case, your output must reflect the username and password which you used.

**Pushing work into GitHub**

We need to understand some commands used with git. Here are some of the common commands

**Creating a Git repository**

git init - initialize an existing directory as a Git repository

**git add**

Adds files in the to the staging area for Git. Before a file is available to commit to a repository, the file needs to be added to the Git index (staging area). You can add specific file buy using:

git add <filenames>

Or you can add all the files in the directory by using:

git add .

**git commit**

Record the changes made to the files to a local repository. For easy reference, each commit has a unique ID and the identifier -m is used to attach a message e.g.

git commit -m “second push”

**git push**

Is used to push the committed changes which are now on the stagging area to the remote repository (GitHub) Once you type this command, an exact copy of the repository will be available in GitHub

git push -u origin main

**git remote**

To connect a local repository with a remote repository. A remote repository can have a name set to avoid having to remember the URL of the repository.

# Add remote repository

git remote <command> <remote\_name> <remote\_URL>

example:

git remote add origin https://github.com/gpdzvapatsva/testing\_repo.git

# List named remote repositories

git remote -v

**git branch**

To determine what branch the local repository is on, add a new branch, or delete a branch.

**EXAMPLE 4.7**

* Create a folder on the desktop and call it git\_test.
* Let us create a C++ project called testing and save it in the git\_test folder
* By default, the main.cpp file shows default code for hello world.
* Close the project and open the git\_test folder. Type cmd in the address bar to open the git\_test in command prompt. There are various ways of navigating. Do the one which works well for you.
* Now initialise your folder to a git repository by typing: git init
* Next add files to git repository by typing : git add .
* Commit changes by typing git commit -m “initial push”

Figure 4.32 shows all the above commands in succession

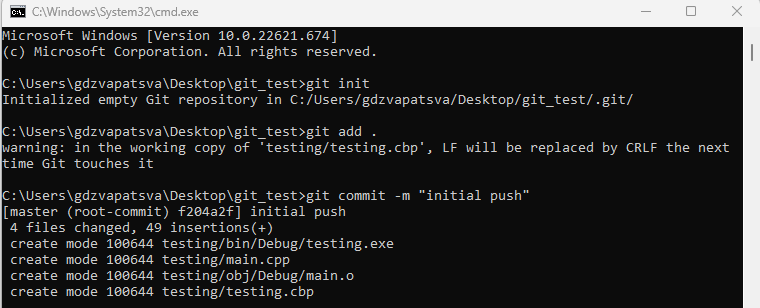


Figure 4. 32: Initialising a repository

* Now open GitHub using the credentials you used to create an account.
* On the left panel you will see a green button inscribed New. Click it and give a name to your remote repository. Let use c++\_hello\_world and hit create repository at the bottom.
* Now you will be presented with several option of pushing your work. For the purposes of this practice- select the option which says: “…or push an existing repository from the command line”. Copy all the three lines together and paste them on the command prompt as shown in Figure 4:33 and hit enter.

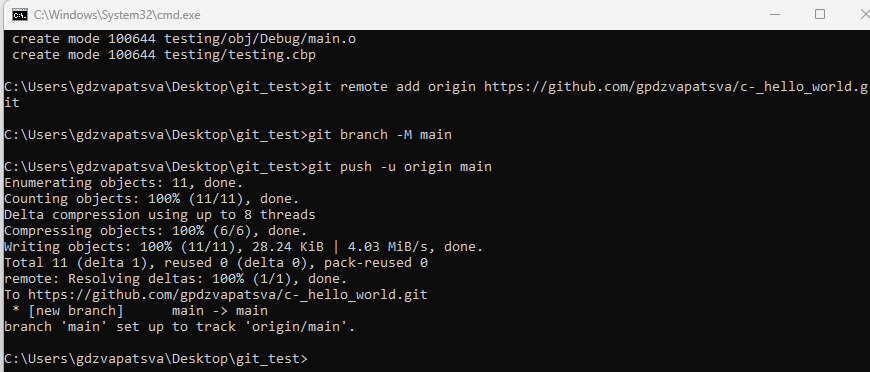


Figure 4. 33: Pushing work into a GitHub

* You will notice that a URL has been created for your repository. You can now share the repository with whoever wants the project.

Congratulations. You have pushed your first project.

* + 1. **Install and configure Git on Raspberry Pi**

Raspberry Pi 4 model comes with Git preinstalled. Simply run the same commands as on Windows machine as shown in Figure 4.34.

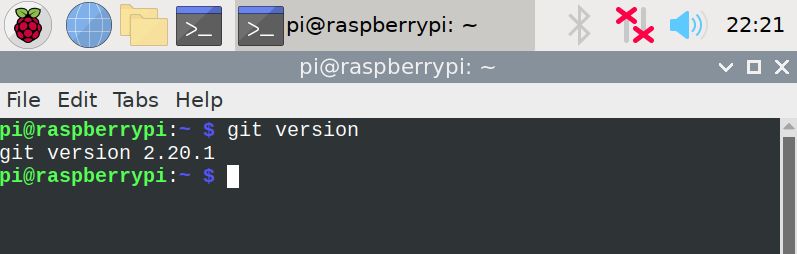


Figure 4. 34:Git of Raspberry Pi 4

For most people, the easiest and the recommended way to install Git is to install it using the [apt](https://linuxize.com/post/how-to-use-apt-command/) package management tool.

Installing Git with Apt

The Git package is included in the Raspbian default repositories. Here are the steps to install git on earlier versions.

**Step 1:** Open the terminal of Raspberry Pi OS, then write the following command:

$ sudo apt update

$ sudo apt install git

**Step 2:**Then write ‘Y’ to proceed further. And wait for some time to install

**Step 3**: Check if Git has been successfully installed by running the following command:

git –version

**Step 4**: Now configure Git using the following command:

git config –global user.name “your name”

git config –global user.email “your email”

* + 1. **List and explain common terminology associated with distributed VCS’s including:**

**Branch**

Earlier in section 4.6.4, we introduced branching. Just a reminder, branching means you diverge from the main line of development and continue to do work without messing with that main line. Developers can continue to work without affecting the main branch and if satisfied then merge to the main branch.

How to create a branch

We are going to use the git\_test repository we created for the hello\_world program. Syntax for creating a branch:

Create a new branch

>git branch <branch\_name>

Let us create a branch called first\_branch

After creating the branch we can check how many branches are on our repository by typing:

>git branch -a

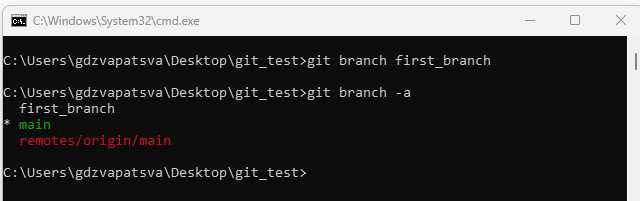


Figure 4. 35:Creating and checking branches

As you can see in Figure 4.35, we created a branch called first\_branch. There was the main branch already and after checking the number of branches we see two branches as output.

**Checkout**

git checkout

To start working in a different branch, use git checkout to switch branches.

> git checkout <branch\_name>

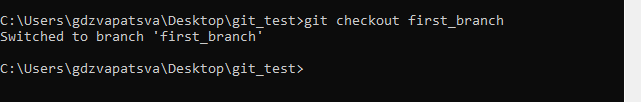


Figure 4. 36:Git Checkout

You can also move to a branch and create at the same time by typing the following command:

git checkout -b <branch\_name>

See Figure 4.37

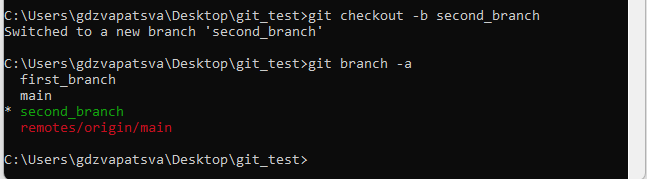


Figure 4. 37:Creating and navigating to branch

To delete a branch, you need to type:

git branch -d <branch\_name>

**NOTE:** To delete a branch, you must navigate to a branch in a higher level. For instance, to delete second\_branch, you must switch to the main branch first.

**Clone**

git clone

To create a local working copy of an existing remote repository, use git clone to copy and download the repository to a computer. Cloning is the equivalent of git init when working with a remote repository. Git will create a directory locally with all files and repository history.

Syntax

git clone <remote\_URL>

EXAMPLE

Lets clone a Python repository I created. The repository name is called <https://github.com/gpdzvapatsva/areaofcircle>

Make sure you navigate to the directory where you want the code cloned into. In our case, let us navigate to the desktop.

C:\Users\gdzvapatsva\Desktop>git clone <https://github.com/gpdzvapatsva/areaofcircle>

Output

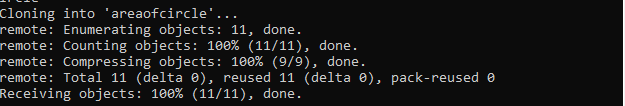


Figure 4. 38:Git clone

Compare your output with that on Figure 4.38 and check if you now have that Python project on to your desktop.

**Stage**

Staging is a step before the commit process in git. That is, a commit in git is performed in two steps: staging and actual commit. staging helps you keep extra local files hanging around.

git-stage - Add file contents to the staging area

**Commit**

git commit

As mentioned earlier, git commit is used to record the changes made to the files to a local repository. For easy reference, each commit has a unique ID. It’s best practice to include a message with each commit explaining the changes made in a commit. Adding a commit message helps to find a particular change or understanding the changes.

Syntax

git commit -m <comment>

git commit is used after adding changes and before pushing the changes to the remote repository. You have seen how we committed the git\_test folder. If we make any changes to the file, we will need to run git add and then git commit.

**Conflict**

When two different branches make updates to the same line in a file, or when a file is destroyed in one branch but changed in the other, a conflict results. For instance, conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it.The likelihood of conflicts occurring increases when working in a team environment Conflicts over merges can be resolved using a variety of tools.

**Head**

In the current checkout branch, the HEAD identifies the most recent commit. Like a pointer to any reference, it. The "current branch" is known as the HEAD. The HEAD is moved to the new branch when you switch branches with "checkout." The **git show head** is used to check the status of the Head. This command will show the location of the Head. Figure 4.39 illustrates the output of git show head.

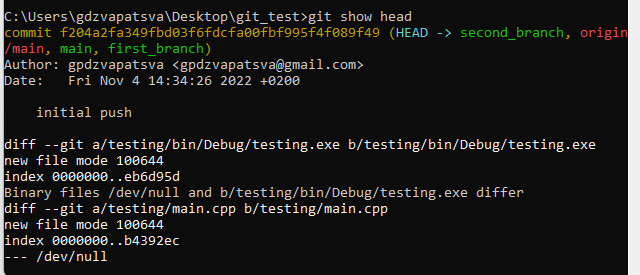
****

Figure 4. 39: Sample output of git show head

**Repository**

Each file in your project, together with its revision history, is stored in a repository. Within the repository, you can coordinate and oversee project activities. To see also if the folder is a git repository, open the folder, you will find another subfolder with extension .git inside.

You typically obtain a Git repository in one of two ways:

* You can take a local directory that is currently not under version control, and turn it into a Git repository, or
* You can clone an existing Git repository from elsewhere.

Git has four main states in which files can be saved: working directory, staging area, local repository and central repository.

* Working directory: where projects are created (code written) and changes are made
* Staging area: where code can be reviews before making a commitment
* Local repository: where you may commit changes to the project before pushing them to the central GitHub repository
* Central repository: where the primary project is stored (a copy is kept locally by each team member).

**Initialise**

A new Git repository is created with the git init command. It can be used to start a fresh, empty repository or convert an existing, unversioned project to a Git repository. This is typically the first command you'll perform in a new project because most additional Git commands are not accessible outside of an initialized repository.

# FORMATIVE ASSESSMENT 4.6 INDIVIDUAL TASK

4.6.1 Define the term version control. (2)

4.6.2 What is the difference between Git and GitHub? (4)

4.6.3 List FIVE benefits come with using GIT? (5)

4.6.4 Write the command used to initialise a Git repository. (2)

4.6.5 What does the command git -v do when typed on the git bash terminal. (2)

4.6.6 Practical

* Create a folder on the desktop and name it “git\_revision”.
* Add a text file called file.txt
* Add the following line of text through the terminal into a file called file.txt

“Working with git and github is saves time”

* Add the directory to git (including the file.txt)
* Commit the changes and add a comment which reads “initial push”
* Create a remote repository and give it the same name as the local repository and push the local folder to remote repository.
* What is the name of the remote repository.

(20)

**Total :35 marks**

# SUMMATIVE ACTIVITY 4.7 INDIVIDUAL TASK

4.7.1 Differentiate centralised version control systems from distributed version control systems. (4)

4.7.2 Define the term source code as applied in computer programming. (2)

4.7.3 List FOUR examples of IDE’s used for C++ programming. (4)

4.7.4 Explain how to display out from a C++ program. (2)

4.7.5 What are the THREE advantages of using GIT? (3)

4.7.6 Execute a Git command to create a copy of the following repository.

https://github.com/gpdzvapatsva/git\_revision.git

Write the command you used. (2)

4.7.7 What does the command git config --list do? (2)

4.7.8 What are the major differences between GCC and G++. (4)

4.7.9 Execute a git command to change the current user registered on your machine to “cplusguru”.

What command did you use to do this. (2)

4.7.10 List THREE common debugging techniques. (3)

**Total :28 Marks**

**References**

<https://education.github.com/git-cheat-sheet-education.pdf>

<https://git-scm.com/>

<https://byjus.com/gate/difference-between-gcc-and-g-plus-plus/>

# Module 5: Introduction to a high-level programming language

By the end of the module, students should be able to:

* Create a new project using an IDE
* Create a new file with extension CPP
* Explain why C++ source files use the .cpp extension and C++ source files uses the .c extension.
* Write the C++ code to display output “Hello World”.
* Compile a C++ application using an IDE
* Explain what type of file is produced by the C++ compilation process
* Show or hide the IDE message window
* Investigate the IDE message window to determine if compilation succeeded
* Investigate the IDE message window to locate line numbers that contain errors
* Build a C++ application using an IDE
* Explain what file type is produced by the C++ build process
* Run a C++ application using the IDE
* List and explain the different parts of a simple C/C++ application
* Explain case sensitivity as it applies to C/C++
* Explain the term free-format language and how that applies to C/C++
* Explain the term block-structured language and how that applies to C/C++
* Explain what a comment is used for in the C/C++ language
* Explain how the compiler will treat a comment
* List and explain two types of comments allowed by the C/C++ compiler
* Discuss the importance of adding comments to a C/C++ application

**Introduction**

Geany is a lightweight, high-performance Integrated Development Environment (IDE) system that supports the most used programming languages, including C/C++, Java, Python3/x, and others. Geany runs on Linux, Windows and macOS and has built-in support for more than 50 programming languages.

**How to install Geany**

**Step 1**: Search for Geany downloads. You can enter the following url:

<https://www.geany.org/download/releases/>

Step 2: Select your operating system. In our case we select windows.



Figure 5. 1: Geany downloads

**Step 3**: Double click the downloaded executable file and click Next on the screen indicated on Figure 5.2.

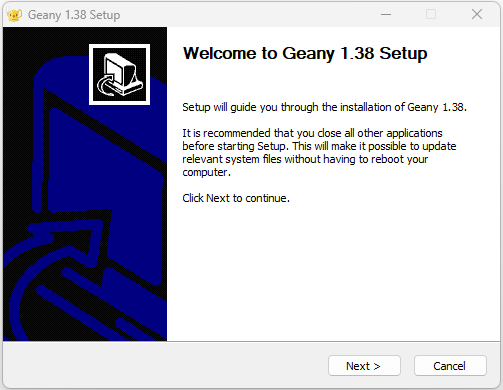


Figure 5. 2: Geany welcome screen

**Step 3**: Accept the license agreement and select full installation

**Step 4**: Select the path from your machine on which Geany must be installed and click next. The screen will look as the one shown in Figure 5.3.

Graphical user interface, text, application, email

Description automatically generated

Figure 5. 3:Choosing installation location

**Step 5**: The last screen will show that Geany has completed the installation process. You can opt to start Geany or show release notes. You will see a screen as the one in Figure 5.4. Click Finish to complete setup process.

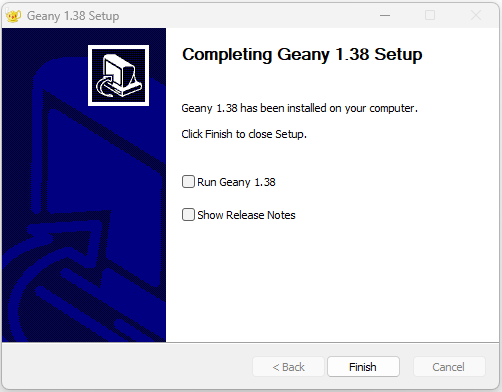


Figure 5. 4: Complete setup

If you do not have the compiler installed you can search for the following address in the browser: <https://jmeubank.github.io/tdm-gcc/download/>

You can then select to download the MinGW which includes GCC C/C++, GNU binutils, mingw32-make, GDB (64-bit), the MinGW-w64 runtime libraries and tools, and the windows-default-manifest package. In our case, you recall we setup our compiler already when we used Code::Blocks. So, you can skip this step.

### Create a new project using an IDE

**Step 1:** Start Geany by tying its name in the search bar or from the task bar alternatively click the Geany icon on the desktop (if one has been created during installation process)

**Step 2:** Click on *Document* from the menu *bar*, select *Set FileType* then Programming Languages and click on *C++ source file.*

**Step 3: By default, you will see some license text in the file called untitled**

### 5.1.2 Create a new file with extension CPP

In section 5.1.1, we learnt how to start Geany program. In this section, we are going to discuss how to create a C++ file with a .cpp file extension.

Step 1: Follow the steps 1 to 3 in section 5.1.1.

Step 2: Select the save icon as indicated in Figure 5.5.

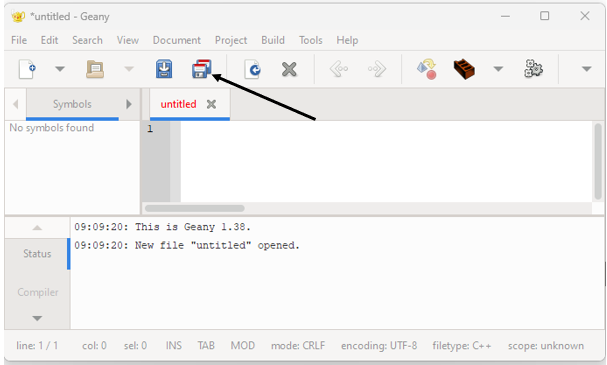


Figure 5. 5:Geany start-up window

Now, you will need to select the path where your program must be saved. You can create a folder where the current program will be saved. In our case, let us create a folder called Helloworld on to the desktop.

Change the file name from untitled to hello\_world.cpp.

If you navigate to the Desktop folder called Helloworld, you will see an image like the one below.

Graphical user interface, application

Description automatically generated

This is a C++ file with extension .cpp.

### 5.1.3 File extension in C and C++

Historically, the first C++ extensions used were .c and .h, just like C. This caused practical issues, particularly with the .c extension, which made it difficult for build systems to distinguish between C++ and C files. Because C language sources typically have the extensions ".c" and ".h," it was common for C++ source files to share the same extensions or use a distinct variation to clearly indicate the C++ code file in the beginning. Today, most C++ implementation files will have the ".cpp" extension and header files will have the ".h" extension (the last one is still shared across most assembler and C compilers).

Other common extension variations for "implementation" code include ".cc", ".C", ".cxx", and ".c++". The same extension variations are used for header files, but the first letter of the extension is usually replaced with a "h," as in ".hh", ".H", ".hxx", ".hpp", ".h++", and so on. For the purposes of this module, we will stick to .cpp

* + 1. Write the C/C++ code to display output “HelloWorld”.

Hello World is commonly used by programmers as the default start program. A "Hello, World!" program is a simple program that displays the phrase "Hello, World!" on the screen.

Using the program called hello\_world.cpp created earlier in section 5.1.2, type the following source code:

**EXAMPLE 5.1**

#include <iostream>

using namespace std;

// main() is where program execution begins.

int main() {

cout << "Hello World"; // prints Hello World

return 0;

}

So let us try to understand the structure of a C++ program. In general, the structure of C++ program is as follows:

|  |
| --- |
| C++ Headers |
| Class definition |
| Member functions definitions |
| Main function |
|  |

Figure 5. 6: General structure of a C++ program

For now, we are not going to dive deeper into the structure. We will try to, compile, and run our project first. We will explain the structure in detail in section 5.2.1. Please note that the Hello world will not display any out even to tell us if there were no errors in the program. The only way to find out will be to compile the source code.

### 5.1.5 Compile a C++ application using an IDE

Compiling a C++ program entails converting the source code (.cpp files) into an executable or library that can run on a specific platform. This process can be divided into three key stages:

* **Pre-processing**-The outputted code will be ready to compile by the end of the pre-processor stage.
* **Compilation-**C++ compilation is a two-step procedure. First, the compiler converts the source code to assembly language. Second, using an assembler, the assembly language is converted into actual machine code. The resulting output is a collection of files known as an object file. The object code file has .obj or .o file extension
* **Linking**-The final stage generates the executable or library.

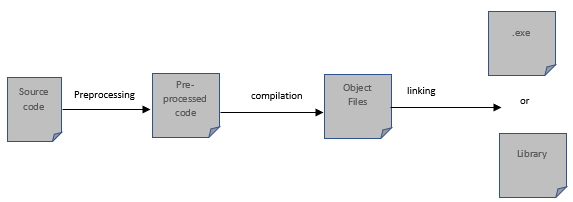


Figure 5. 7: C++ compilation

### 5.1.6: Type of file is produced by the C/C++ compilation process

Let us look in the folder where we saved our project created in 5.1.4. Before compilation, there is one file called hello\_world.cpp

Steps to compile our project

**Step1:** Select *Build* from the menu and select compile. You can click the icon

If you navigate in the folder where we saved our program, you will now see two files as shown in Figure 5.8.

Graphical user interface, text, application

Description automatically generated

Figure 5. 8:C++ Output files

As illustrated in Figure 5.8, the first file is the source code file. The second file (hello.o) is the object file with a .o extension. But for us to see the results, we need to build the program in order to generate an executable file. Executable file will be generated when we select build option from the Build category.

### 5.1.7 Show or hide the IDE message window

Figure 5.11 shows the message window in the Geany IDE.

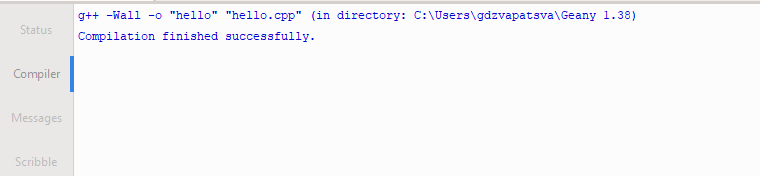


Figure 5. 9:Message window

An optional message window in Geany shows the following tabs:

* Status - A list of status messages. Here is a sample of the status message:

16:34:43: File C:\Users\gdzvapatsva\Geany 1.38\test.cpp saved.

* Compiler - The output of compiling or building programs. In my case, this is what the compiler section of Geany is showing:
* Messages - Results of 'Find Usage', 'Find in Files' and other actions
* Scribble - A text scratchpad for any use.

**Steps to show/hide the message window**

To display the message window, select *View* and check the *Show Message Window* box.

To disable the message window, go to the *View* menu and uncheck the *Show Message Window* checkbox.

5.1.8 Check if compilation succeeded on Message Window

When you compile the C++ program, the Message Window will display if the process was successful or failed. This will be displayed in form a message as shown below:

g++ -Wall -o "hello" "hello.cpp" (in directory: C:\Users\gdzvapatsva\Geany 1.38)

Compilation finished successfully.

5.1.9 Investigate the IDE message window to locate line numbers that contain errors

When using Geany IDE, and your code has errors, these are shown during the build process.

Consider the following code.

EXAMPLE 5.2

// Your First C++ Program

#include <iostream>

using namespace std;

int main() {

<< "Hello World!";

return 0;

}

/\*

Compiling the program causes errors to be displayed in the Message Window. Figure 5.10 shows the errors in the message window:

Text

Description automatically generated

Figure 5. 10:Error messages from the program

As seen in Figure 5.10, Geany does show the line numbers and the suggestion of what the error is about. In the given scenario, it is pointing that an expected primary expression before’<<’ is expected. So, we just type in cout. Recompile, build, and run the program again and the error is not there anymore.

### 5.1.10 Build a C/C++ application using an IDE

Now let us do the practical on performing build on a C++ using the Geany IDE.

**Steps**

Open the hello\_world.cpp file created in section 5.1.2. Just save and compile again in case there were some changes to the file.

**Step1:** Select *Build* from the menu and select compile. You can click the icon

****Step 2:** Select *Build* from the menu bar and click *build.* Alternatively click on the

### 5.1.11 Explain what file type is produced by the C/C++ build process

Let us examine our HelloWorld folder on the Desktop.

Graphical user interface, text, application

Description automatically generated

Figure 5. 11:Build result of C++ program

As you can see in Figure 5.11, we now have an executable file hello\_world with and extension exe. This is the file which we can run to see our output of the program. For now we are going to use an IDE to complete the running of the program.

### 5.1.12 Run a C/C++ application using the IDE

Remember the executable file does not serve its purpose if it has not been run. Users need to see the output of the program and this is achieved by running the program.

Step 1: To run the file, repeat compiling and building in case there were some changes.

Step 2: Click on the  icon to run the program. The output will be as shown in Figure 5.12

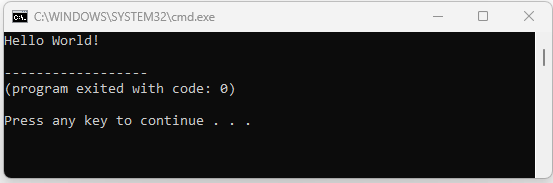


Figure 5. 12:C++ Output

Once you press any key the output disappears. We can also do the same processes using a command line.

Running a program allows users to see the output of the program. When using Geany IDE, running a program is very easy. You will need to follow steps to compile and build the application first. Once done, you need to select Build from the menu bar and click on execute. This will trigger the executable file to be activated and output to be displayed on the command prompt.

It is important to note that, developers can opt for the out to be displayed on different interfaces depending with choice of the organisation. Some of the common interfaces are:

* Command Line Interface (CLI)
* Web Based Interface (WBI)
* Graphical User Interface (GUI)
* Voice user interface ([VUI](https://www.techtarget.com/searcherp/definition/voice-user-interface-VUI))
* Touch user Interface

We briefly discussed CLI and GUI in section1.2.3. We are going to explain them just to keep you up to speed with user interfaces.

**Command Line Interface (CLI)**

Command Line Interfaces accept keyboard commands as input; the commands entered at the command prompt are then executed by the computer. Command-line interfaces can be found in the MS-DOS operating system and the command shell in the Windows operating system. For the purposes of this module, we will stick to command line interface.

**Graphical User Interface**

A graphical user interface (GUI) is a graphical (rather than purely textual) user interface to a computer. This is most common for desktop platforms. Users interact with the program through clicking graphical.

**Web Based Interface**

This is common for online programs where users interact with the program through web browsers. Most organisations have resorted to this interface to allow their users to access their programs from any geographical location.

**Voice User Interface**

Voice user interface (VUI) is a type of speech recognition technology that allows people to interact with a computer programs by speaking commands.

**Touch User Interface**

This graphical in nature but response to user touch for input. This is now implemented in most banks as developers are becoming more conscious with the aspects of human computer interaction.

**How to create a C++ file from Notepad and compile from Command Prompt**

**Step 1:** By now you know how to check if the compiler is installed. If you have forgotten, its pretty simple. Simply open the command prompt and type the following command :g++ -- version.

**Step 2:** Create a folder on the Desktop and give it any name for example:

C++\_Programming. You can do all these using the following commands:

C:\Users\gdzvapatsva>cd Desktop

C:\Users\gdzvapatsva\Desktop>mkdir C++\_Programming

C:\Users\gdzvapatsva\Desktop>notepad

C:\Users\gdzvapatsva\Desktop>

**Step 3:** When Notepad opens, save the file as hello\_world.cpp. Type the following code:

#include <iostream>

using namespace std;

int main (){

std::cout<<"Hello programmers";

return 0;

}

Make sure you change the *Save as type* to All types

**Step 4**: If you open the C++\_programming folder, you will notice our source file hello\_world.cpp is there. So navigate to this path in command prompt.

**Step 5**: Type the following command:

g++ hello\_world.cpp and hit enter

**Step 6**: Type a in the command prompt

The output will be as follows:

Text

Description automatically generated

Figure 5. 13:Command Line output for the compilation

So the program has been compiled and run. If you go and check the folder with our program, you will see an executable file call a. a is a default filename assigned to the executable file.

**Note:** You can also assign a name of your choice to the executable file instead of the default file name a.exe. To do so go to step 5 and modify it as follows:

g++ hello\_world.cpp -o hello.exe

Now you can go back to command prompt and type hello.

The output will still be same as in Figure 5.9.

The -o flag indicates that you are assigning a file name.

**eLink**

<https://www.youtube.com/watch?v=GxHLErBLgI8>

# FORMATIVE ASSESSMENT 5.1 INDIVIDUAL TASK

5.1.1 What is a high level programming language. (2)

5.2.2 Define the term IDE and give two examples of IDE’s which can be used for C++.(2)

5.2.3 What is the most common file extension for C++ source file. (2)

5.2.4 With the aid of the diagram, explain how C++ files are compiled. (4)

5.2.5 Consider the following C++ program

#include <iostream>

using namespace std;

int main (){

/\*This is the main block

I am learning about multiline comments

\*/

cout<<"my name is Donald";

return 0;

}

Identify the following parts on the program:

* Preprocessor directive
* namespace
* Header file
* Comments
* Main function
* Return statement (7)

5.2.6 Create a file called exams using notepad and save it in exams folder on the Desktop.

Enter the following C++ code:

#include <iostream>

using namespace std;

int main (){

/\*This is the main block

I am learning about multiline comments

\*/

cout<<"Compiling C++ using command line";

return 0;

}

Write a command to compile the source file and produce an executable file called exams1.exe.

(10)

5.2.7 List THREE commands you need to select to compile and run C++ code on an IDE such as Geany. (3)

5.2.8 Differentiate compiling and building in C++. (4)

**Total: 34 marks**

## 5.2 C++ application structure

### 5.2.1 List and explain the different parts of a simple C/C++ application

So far, we have learned how to create a basic C++ program. Next, we will identify the different parts of a simple C++ program. A C++ program is structured in a specific and unique way. A program in C++ is divided into the three sections listed below:

1. Pre-processor directives
2. Namespace
3. Main function
4. Body of main function
5. Opening and closing braces

Figure 5.14 shows the different parts of a simple C++ program

Diagram

Description automatically generated with medium confidence

Figure 5. 14: Structure of a C++ program

So let us explain each of the identified parts and explain them

#### 5.2.1.1 Headers

It is important for us to define preprocessor directives first when discussing the structure of a C++ program. Lines beginning with a hash sign (#) are directives read and interpreted by what is known as the preprocessor. An example of preprocessor directive is the #include. These are typically used to make source programs easy to change and easy to compile. The [preprocessor](https://en.wikipedia.org/wiki/preprocessor) is either a separate program invoked by the [compiler](https://en.wikipedia.org/wiki/compiler) or part of the compiler itself. Before the compiler attempts to compile the resulting source code, the preprocessor performs intermediate operations that modify the original source code and internal compiler options.

Headers or .h files are libraries of code you may insert in your program by including them through referencing them after the preprocessor directives. In C++, all the header files may or may not end with the .h extension.

Syntax of Header File in C/C++

#include<filename.h>

Example:

#include <iostream>

Here are some common examples of headers:

* #include <iostream> input/output interaction with the program
* #include<time.h>(Time header)- Use date and time functions such as setdate() and getdate() (). To change the system date and obtain the CPU time.
* #include<fstream.h> (File stream) – Used to control the data to read from a file as an input and data to write into the file as an output.
* #include<math.h> (Math header )- Perform mathematical operations like sqrt() and pow(). To obtain the square root and the power of a number respectively.
* <ctype.h>(Character type header)- Use character type functions such as isaplha() and isdigit() ().
* #include <string>

C++ programs must include the header file <iostream>, which stands for input and output stream and is used to take input using the "cin>>" function and display the output using the "cout" function.

#### 5.2.1.2 Name space

A namespace is a declarative region that defines the scope of the identifiers (the names of types, functions, variables, and so on) contained within it. Namespaces are used to organize code into logical groups and to prevent name collisions, which can occur when your code base includes multiple libraries. All identifiers at namespace scope are unqualifiedly visible to one another.

All C++ standard library types and functions are declared in the std namespace or namespaces nested inside std. The using pre-appended to the namespace instructs the compiler that the following code will use names from the specified namespace. A namespace definition begins with the keyword namespace followed by the namespace name as follows:

**EXAMPLE 5.3**

#include <iostream>

using namespace std;

// main() is where program execution begins.

int main() {

std::cout << "Hello World"; // prints Hello World

return 0;

}

Once declared in the top block, you can leave it out in the function block.

Programmers can also define their own namespaces as shown below.

**EXAMPLE 5.4**

#include <iostream>

using namespace std;

// first namespace

namespace ncv\_level\_two

{

void display()

{

cout << "I am in level two" << endl;

}

}

// second namespace

namespace ncv\_level\_three

{

void display()

{

cout << "I am in level three" << endl;

}

}

using namespace ncv\_level\_three;

int main ()

{

// calling from ncv\_level\_three namespace.

display();

return 0;

}

Output

Graphical user interface, text

Description automatically generated

If we change the namespace we are calling in int main() function to ncv\_level\_two, the output will be: “I am in level two”. So, we used the namespaces to arrange our code nicely and separate code. Although we used the same function name *display(),* our program could identify which one we were referring to at any particular point.

How about if we wanted to call the two display() in the different namespaces. This is easy. We could modify our main function as follows:

**EXAMPLE 5.5**

int main ()

{

// calling all the code in different namespaces

ncv\_level\_two::display();

ncv\_level\_three::display();

return 0;

}

Output

Graphical user interface, text

Description automatically generated

In this program we also introduced a new word called *endl. endl* Inserts a new-line character and [flushes](https://cplusplus.com/basic_ostream::flush) the stream. If we remove it, you will notice that our output is printed in the same line. Its behaviour is equivalent to calling (‘\n’) for inserting new line.

#### 5.2.1.3 Main

This line starts the function declaration. A function is essentially a group of code statements that are given a name: in this case, this gives the group of code statements that follow the name "main." Functions will be discussed in detail in a later chapter, but in general, their definition begins with a type (e.g int), a name (main), and a pair of parentheses (()), optionally followed by parameters.

The main function is a special function in all C++ programs; it is called when the program is executed. All C++ programs begin execution with the main function, regardless of where the function is located within the code. The main function has two brackets in front of it to all adding parameters. We learned about parameters in level 2. We will not discuss them in detail here but in later chapters. After the brackets we have opening brace and closing brace. The code will be placed inside the braces. { signifies the start of a block of code, ​and } signifies the end.

int main() {

}

#### 5.2.1 return

Return statement (s) terminates the execution of a function and returns control to the calling function (or, in the case of main function, transfers control back to the operating system). Execution resumes in the calling function at the point immediately following the call. In the case of our example shown in section 5.2, the return is 0 since the default return value of C++ program is an integer unlike in C programming where the main{} function returns void.

### 5.2.2 Explain case sensitivity

C++ is a case sensitive programming language so; all the keywords must be in lowercase. Case sensitive means that the uppercase and lowercase letters are considered differently. Variable names intNumber1 and intnumber1 are different.

Here is an example of an error message the compile will generate if you try to treat same words differently.

**EXAMPLE 5.6**

Graphical user interface, text, application

Description automatically generated

Figure 5. 15: Case-sensitivity

When you compile the above program, an error message will be displayed in the message window as shown in Figure 5.15.

Graphical user interface, text

Description automatically generated

Figure 5. 16:error messages for case-sensitive

The program generated error because of two intNumber1(i.e line 5and 7) declared. However, the compiler did not see an error between *intNumber1* (line5) and *intnumber1* (line 6).

### 5.2.3 Explain the term free-format language and how that applies to C++

A free-format language is a programming language in which character positioning on the page in program text is irrelevant. Program text does not need to be placed in specific columns. C++ is a free-format programming language. The benefit of free format language is that programmers can type in code without worrying about line breaks. Here is an example of a code snippet.

Logo, company name

Description automatically generated

Figure 5. 17: Free-formatting

Line 7 is a continuation of line 6 and the keyword cout is way in the middle column. When compiled and executed, the program runs fine without generating any error message. Other languages such as Python will generate an “IndentationError: unexpected indent”.

### 5.2.4 Explain the term block-structured language and how that applies to C++

**DEFINITION**

A block-structured programming language is one that allows the creation of blocks, including blocks nested within other blocks.

A block consists of a sequence of statements and/or blocks, preceded by declarations of variables. Block structured languages define blocks of scope using some sort of delimiter pair for e.g. braces in C++ { }. A code block can be the body of a function, or it can be controlled by conditional execution (if statement) or repeated execution (while statement, for statement, etc.) We will explain what conditional statements are in later chapter 8 and 9. Consider the code shown in example 5.7

**EXAMPLE 5.7**

#include <iostream>

using namespace std;

namespace addition

//This is a block of code

{

void add\_numbers(){

int age=19;

cout<<"The age is "<<age;

}

}

int main ()

//This is the main block

{

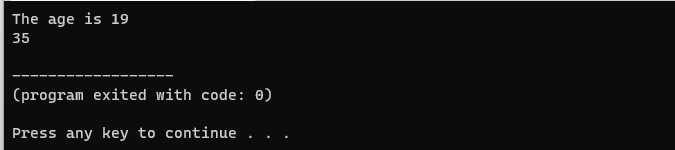
//calling a block of code

addition::add\_numbers();

cout<<endl<<7\*5 ;

}

Output



You can see from example XXX that the code is split into structured block. Entry to a block is seen with braces. In the example, there are two blocks of code: the main block and the addition block. If we write the following statement in the main block:

cout<<age;

We will get an error because it was declared outside the main block and its scope not recognised.  The scope of age is confined to the block in which it has been declared only.

### 5.2.5 Explain what a comment is used for in the C++ language

Comments in C++ are hints that a programmer can add to make their code easier to read and understand. C++ compilers ignore comments completely and treats them as white space. Comments are normally used to annotate code for future reference.

**Syntax of a comment**

//declare variable for first number

Int intnumber1=10;

You can also create a comment that displays at the end of a line of code. But generally, its a better practice to put the comment before the line of code.

### 5.2.6 Explain how the compiler will treat a comment

C++ compiler ignores comments. Preprocessors in C++ compilers analyze each character in a source file sequentially, before any other compilation phase. It ignores the beginning of a comment, either /\* or //. When it does, it continues to consume source file characters, throwing them away until it encounters either a \*/ or the end of line character (for single-line comment). Regardless of how the comment ends, the preprocessor continues processing subsequent characters normally until another comment begins. The preprocessor typically replaces the entire comment with a single whitespace character (e.g., space, tab, newline). In some cases lexical analyser are responsible for detecting and throwing away comments. It is the lexical analyzer's job to break down the source code into tokens, which are then fed into the parser. When the lexical analyser meets // or /\* \*/, it sends nothing to the parser and continues to do so until it finds valid token.

### 5.2.7 Two types of comments allowed by the C++ compiler

There are two ways to add comments to code:

// - Single Line Comments

/\* \*/ -Multi-line Comments

**Single Line comments**

EXAMPLE

In C++, any line that starts with // is a comment as shown in Figure 5.18.

A picture containing graphical user interface

Description automatically generated

Figure 5. 18:Single-line comments

Line 5 and line 7 are never executed when the program is compiled and build.

**Multi-line comments**

In C++, any line between /\* and \*/ is als**o a comment**. For example,

Graphical user interface

Description automatically generated with medium confidence

Figure 5. 19:multi-line comments

Comments can also be used to disable code to prevent it from being executed. As an example, if we encounter an error while running the program, we can use comments to disable the error-prone code from executing; this can be useful for debugging. Comments should describe what the program does at a high level. Reiterating something that is obvious is not a good idea.

### 5.2.8 Importance of adding comments to a C++ application

Comments are import in programming only to the programmers. Users will never see the program comments as they are not executed by the compiler.

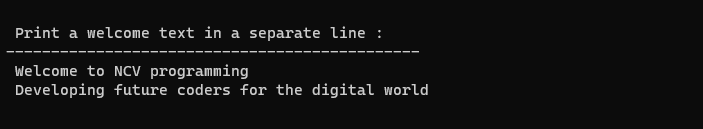
**Importance of comments**

1. Saves time- new programmers who join the organisation can quickly understand the code segments by reading comments
2. Comments helps to identify important blocks of code easily
3. Comments help to understand why programmers used a certain way to get a solution rather than the other option
4. Comments help to add description and clarity of what certain blocks of code do by adding context

# FORMATIVE ASSESSMENT 5.2 INDIVIDUAL TASK

5.2.1 Explain each of the following terms as found in C++ program. (6)

5.2.2 Write a program in C++ to print 3 lines of text each in its on line. The output must look as follows.



(8)

5.2.3 Consider the following C++ program which is supposed to add two numbers hardcoded in the program by the programmer making use of user defined namespaces.

using namespace std;

int main()

{

addition:add\_numbers;

}

namespace addition

//This is a block of code

{

void add\_numbers(){

cout << " The sum of 29 and 30 is : "<< 29+30 <<"\n\n" ;

}

}

(5)

5.2.4 State whether the following statements are true or false.

a) Since C is a subset of C++, all C programs will run under C++ compilers.

b) By merely looking at one or two lines of code, we can easily recognise whether a program is written in C or C++.

c) The main {} function in C++ and in C are the same. (3)

5.2.5 What are comments in C++? (2)

5.2.6 List TWO ways of adding comments in C++ program. (4)

5.2.7 Explain the term free-format language and how that applies to C++ . (2)

**Total:30 Marks**

# SUMMATIVE ACTIVITY 5.3 INDIVIDUAL TASK

5.3.1 Outline the steps to create and run a new C++ project called addition using Geany. (8)

5.3.2 C++ is case sensitive. True or False (1)

5.3.3 Consider the code below and answer the associated questions.

#include <iostream>

using namespace std;

namespace languages

//This is a block of code

{

void display(){

cout << " C++ is an extension of C ";

}

}

namespace jobs

{

void display(){

cout << " I want to work as a backend developer";

}

}

int main(){

}

Complete the given code so as to produce the following output.



Hint: You have to make all namespaces available in the main function. (5)

5.3.4 At which stage is the object file produced in C++. (2)

5.3.5 Explain the following types of categories of errors. (4)

5.3.6 Define block structured language with examples. (4)

5.3.7 List FOUR importance of adding comments to the code. (4)

5.3.8 What is meant by the term breakpoint as applied in debugging.

5.3.9 With the aid of an example, explain what a syntax error is in C++. (3)

**Total :31 Marks**

# References

<https://learn.microsoft.com/en-us/cpp/cpp/namespaces-cpp?view=msvc-170>