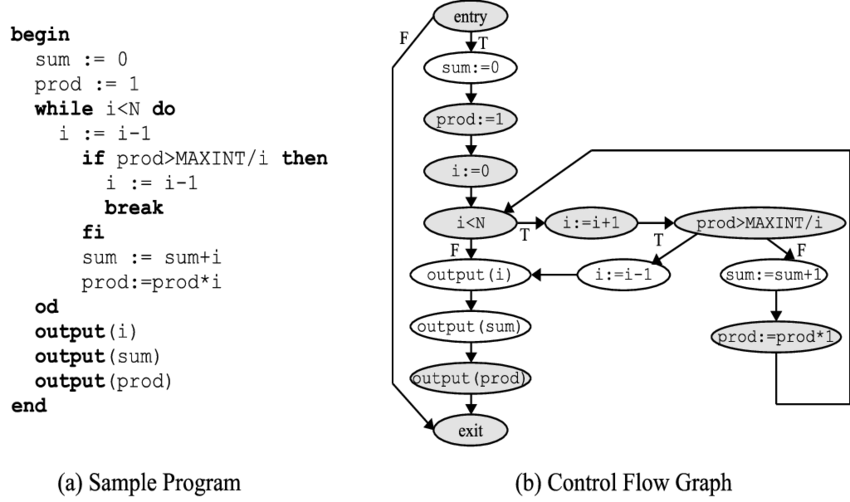
LO2: Understand how programming is implemented.

*2.1 Interpret variables within programming languages.*  
*2.2 Interpret common programming control structures that are used when developing code.*  
*2.3 Describe the use of common data structures.*  
*2.4 Describe how algorithms are used in programming.*  
*2.5 Describe how to test and debug programs.*

Don’t forget images!



# Variables

Variables are used in computer programming to identify data stored in memory. A simple identifier is used to represent the variables according to the rules of the language used. Variables exist within a scope which can be either global or local. To be accessible a variable must be within scope of the calling code, and when a variable of out of all scope, it can be destroyed, and the used memory be made available again during a process called garbage collection.

Variables can hold a range data and can be weakly typed where the same identifier can be used for different types of data or strongly typed where only a single type of data may be stored. Data Types can include numeric data (integers or floating-point numbers), strings, dates or objects but each language will have its own list of available data types.

## Flow Control

Although single sequence programs (where one ordered set of instruction are executed) are common, most programs will require control of the flow of logic, executing blocks of code depending on data stored in variables.

A blue rectangle with white text

AI-generated content may be incorrect.

Figure Normal sequential control flow in imperative/procedural programming paradigms.

Most flow logic can be broken down into two types:

**Conditional Statements** – these evaluate one or more logical statements and execute different blocks of code depending on the results. For example, an If statement or a switch structure. A conditional statement is also known as branching statement as it may change the flow to a different branch of an execution tree.

A diagram of a diagram

AI-generated content may be incorrect.

Figure Conditional control flow

Here is a simple example:

IF (colour=red) THEN

PRINT “You Choose Red”

END IF

**Loop Statements** – Loop statements are used to repeatedly call blocks of code controlled by the result of a logical statement. Examples include Loop and Do..While.



Figure Looping Flow Control

Here is a simple code example:

count = 0

DO

PRINT count

count = count + 1

WHILE (count < 10)

### Branches

Fsdfsfsdfsd

### Iteration

In your report, create a new section called ‘Implementation’.

In this section, explain with examples the following concepts:

* Variables: integer, float, Boolean, string, character.
* Simple programs that show sequencing
* Use of branches (if...then...else, elseif, case/switch
* Iteration (for next loops, while repeat etc)

# Data Structures

Add a new section called ‘data structures’ and then explain, with examples, the following:

* Arrays
* Linked lists
* Stacks
* Queues

## Arrays

Gsdgdsgsdffdsfdsff

hipHipArray = [“hip”,”hip”];

## Linked Lists

Fdsfdsfdsfdsfsd

https://www.linkedin.com/pulse/singly-linked-lists-understanding-real-life-examples-infant-regan-m7vnc/

https://www.geeksforgeeks.org/linked-list-implementation-in-c-sharp/

**Example of Use: Undo Feature in Apps**

Many applications, like text editors, use singly linked lists to implement the "Undo" feature. Each action is stored as a node in the list, and when you undo, the program navigates back through each node sequentially. This approach keeps memory usage efficient and ensures that actions are easily reversible.



## Stacks

fdsfdsfsdfs

## Queues

fdsfsdfsdf  
Algorithms

An Algorithm is a list of rules to perform a specific task. They can be represented in a number of different ways from simple textual descriptions, flowcharts, pseudocode to complex mathematical notations.

For example, here is a pseudocode representation of a simple algorithm:

**Algorithm** LargestNumber

Input: A list of numbers *L*.

Output: The largest number in the list *L*.

**if** *L.size* = 0 **return** null

*largest* ← *L*[0]

**for each** *item* **in** *L*, **do**

**if** *item* > *largest*, **then**

*largest* ← *item*

**return** *largest*

Algorithms can be written for any finite, repeatable solution. Pre-defined algorithms can be used for common problems, or developers can write them to solve one-off problems in an application. Both Sorting data and Searching data can use common algorithms to provide solutions that work and have known levels of efficiency and performance.

## Sort Algorithms

A common task in computing is sorting data into an order such as largest to smallest or alphabetically. Although developers can write their own algorithms when needed, several sorting algorithms exist.

**Bubble Sort** – works by swapping adjacent values (bubbles) until they are in the right order. This is known as a brute force approach.

**QuickSort** – QuickSort is an efficient sorting algorithm. It uses the divide-conquer approach to split the array into sub-arrays that is recursively called to sort the elements. A ‘pivot’ element is selected from the array and partitions the array into two sub-arrays, according to if they are smaller or larger than the pivot. The sub arrays are then sorted recursively until all the data is sorted.

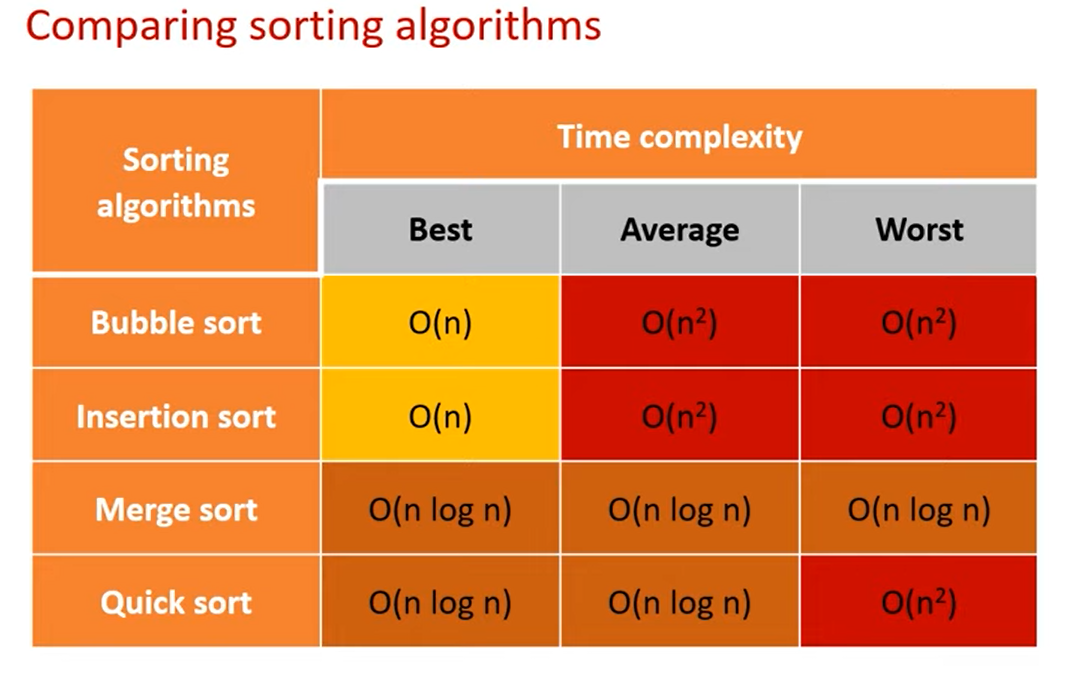
**Merge Sort** - A Mergesort works by applying the divide and conquer approach. The sort begins by breaking the dataset into individual pieces and sorting the elements within a piece. It then merges the pieces to ensures that it has sorted the merged piece.  
The sorting and merging continue until the entire dataset is again a single piece.

‘Big O Notation’ is used to classify algorithms according to the processing time and memory requirements.

A chart with different colored lines

AI-generated content may be incorrect.

Algorithms can be compared by looking at the best, average and worst performance on an algorithm.



Code and illustrations

## Search Algorithms

Write a simple one

Compare some common ones : complexity, effiency, speed

Code and illustrations

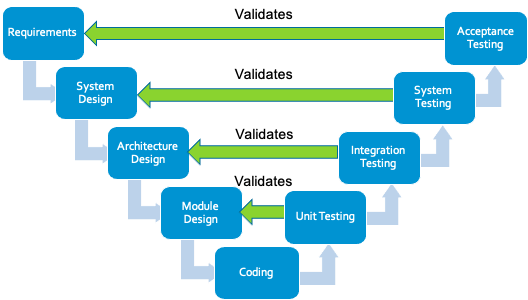
# Debugging

Explain the concept of testing and, using an example program you have written, show how a software program can be tested in different ways such as:

* Use of debugging tools, break points, steps

Watches, conditionals, call stack

# Testing

The aim of software testing to validate that the products we write are of high quality, match the requirements of the customer and perform its within acceptable performance metrics.

There are multiple levels of testing, each primarily targeting different elements of the design and requirements of the application. Some methods are used by developers during the coding process such as Unit Testing, some by professional testers such as System Testing and others by customers such as acceptance testing.

Modern IDE’s will provide tools to help with software testing with automated tools proving feedback as you code, AI tools hunting for anti-patterns (known bad or depreciated coding templates) and analysis tools for performance and code coverage.

## Black Box and White Box Testing

The terms ‘Black Box Testing’ and ‘White Box Testing’ are used to describe two complementary methods of testing. The two methods differ in key points but together can provide coverage of many required testing processes.

**White Box Testing**

* Tester knows about the implementation used.
* Carried out by developers
* Test the structures and logic are implemented correctly and efficiently
* Improves code quality and all code paths tested
* Examples of White Box testing methods include:
  + **Static Analysis** – analyse code without executing it to find syntax errors, anti-patterns. For example, IntelliSense checking for the correct number of arguments to a function call.
  + **Dynamic Analysis** – analyse code performance, behaviour and memory usage while running. For example, testing that memory usage does not soar and exceed requirements during a long process.
  + **Statement Coverage** – ensures every line of code is executed at least once during testing. Also known as Code Coverage.
  + **Branch Testing** – ensure that all possible branches of code are tested.
  + **Path Testing** – checks that all paths through the code are followed.
  + **Loop Testing** – tests loops in code with different conditions to catch error is logic and avoid infinite loops.

**Black Box Testing**

* Performed without access to the code.
* Validates that specified inputs provide specified outputs.
* Carried out mostly by testers
* Examples of Black Box testing include:
  + **Equivalence Partitioning** - Here, possible inputs are group into several partition, where all the members of a partition will be treated the same by the program. Testing one value from the partition is usually enough to cover all scenarios. For example, an age testing routine can be grouped into ‘Too Young’, ‘Acceptable’ or ‘Too Old’.
  + **Boundary Value Analysis** – Frequently we can find errors when data crosses over the values that will change behaviour, so we test values one the boundary and one both sides of the boundary. For example, for a routine checking if an account balance allows a transaction, we would test balances just below, the same as and just above the value of the transaction.
  + **Decision Table Testing** - A decision table is used to represent and test different combinations of inputs and predicted outcomes. This method is effective for testing systems that involve several conditions and actions.
  + **State Transition Testing** - Tests the behaviour in various states and transitions between them. It ensures that the system functions properly when transitioning from one state to another. For example, when a user logs in or out of a system.
  + **Use Case Testing** - Focuses on validating the functionality of the system based on user interactions described in use cases. It ensures that the system meets the requirements of each use case.
  + **Error Guessing** - Relies on the tester’s experience to guess where errors might occur based on common mistakes, past experiences, and known problem areas. For example, entering ‘fish’ into an email field, or a negative value into a loop counter.

## Alpha and Beta Testing

Alpha and beta testing share a common goal of identifying issues before product release. However, they differ in keyways:

**Alpha Testing**

* happens in-house,
* performed by developers or testers,
* in a controlled, simulated environment
* finds bugs and issues early in the development cycle

**Beta Testing**

* happens on customer systems,
* involves real users,
* in real-world (uncontrolled) conditions
* ensures the product matches user needs for functionality and performance.

Alpha Releases are initial versions of the software, used internally by developers and testers to spot and correct bugs. Beta Releases are shared with a select audience outside the development team for further testing and feedback.