**LO1:  Understand programming languages.**

# Procedural Programming

**Procedure Programming** is a programming paradigm that revolves around the concept that programs are a sequence of instructions to be executed. There is also a focus on splitting a program into procedures.

Some of the earliest procedure programming languages were Fortran (1956) and Algol (1958), followed by BASIC (1964), Pascal (1970) and C (1973).

Procedure languages use procedures which are logical blocks of code which can be invoked from other places in the code. These procedures can accept arguments and return values to the caller. Variables declared within each procedure are scoped only to the code within the procedure, but it can also access and modify variables which as within the global scope of the program.

Procedure languages follow structured programming practises where control flow is mandated by the use of reserved words such as **IF**, **WHILE** and **FOR** to define blocks of code. For comparison, non-structured languages (such as assembly) use **GOTO** or branch tables for this purpose.

Procedures callable units of code with a defined behaviour, a defined interface and can be called from other places in the code multiple times. This allows for code re-use, modularity and ease of comprehension. Some programming languages make a distinction between procedures that return a value (functions) and those that do not (subroutine, method or procedure). Others only use the term function whether they return a value or not.

Reusable code can further be grouped into Libraries, typically group together for a purpose, that can be used (or ‘imported’) by any compatible program. For example, a library could provide networking functionality or provide graphical functions. Typically, libraries are used without the need to access the source code behind the library provided by pre-compiled binary code.

Procedures allow for both passing parameters in and retrieving values from it. The exact methods use will depend on how the language is defined, but typically parameters are passed into a procedure by enclosing them as a comma separated list in brackets after the procedures name. If the procedure returns a single value, this is often directly assigned to a variable by use of an equals sign. For example:

**volume = CalculateVolumeOfCube(100, 200,300)**

Here a procedure called CalculateVolumeOfCube is called with three parameters (100, 200, 300) and the returned valued is stored in a variable called volume.

# Debugging Code

Debugging Code is the process of removing errors (called bugs) from your code. Errors can be divided into two main classifications – Syntactic Errors and Logical Errors.

**Syntactic Errors** are coding errors created by not correctly following the syntax of the programming language in use. They will prevent the code from running or compiling and can be caused by something as simple as mismatched quotations or brackets or even a missing semi-colon.

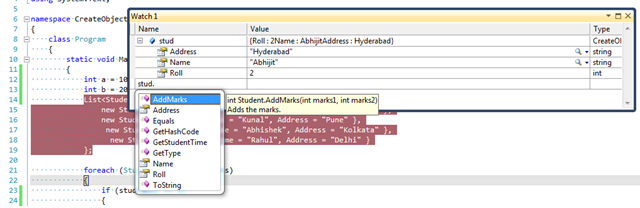
**Logical Errors** are coding errors where code is syntactically correct cause the program to do the wrong thing or generate Exceptions. An exception is how a program reports unexpected or illegal behaviour. For example, an infinite loop could generate a StackOverflowException, or the wrong variable could be passed into a procedure.

There are many tools that developers can use to help find and remove bugs. Often provided as part of an Integrated Development Environment (IDE) or simply as separate tools to be used as needed.

**Logging** allows programs to out messages to some external system (such as a database or a text file). These messages called logs can provide essential information to find out what went happened within a program, even while the program is deployed on the client’s system and far away from a programmer. When an error occurs, the logs can be sent to the programmer to help them identify and correct the error. They can also be used to audit interactions with a system in (for example) a security log.

**IntelliSense** can be provided by a code editing program to highlight the correct syntax for operations such as function calls. By providing the information directly as the code is created IntelliSense greatly decreases the chance of errors. IntelliSense can also incorporate syntax checking which validates the code as it is written. Earlier tools such as ‘lint’ provided syntax checking, leading to this functionality sometimes being called a ‘linter’ or ‘linting’.

A **Debugger** will also allow a developer to step through executing code line by line. A Debugger is an additional piece of software that examines a running program in real time. Use of breakpoints will cause the debugger to pause execution of the code and allow inspection of variables, a list of executed methods (an execution stack). The programmer may then be able to modify these variables to test different scenarios to reproduce or help correct an error.

Debuggers also allow for Watches where the value in a variable is highlighted and may be set to trigger a breakpoint when it changes or reaches a specific value.

Debuggers can also be used remotely to examine code running on other computers either locally or remote over the internet.

# Object Orientated Programming

**Object-oriented programming** (**OOP**) is a programming paradigm based on the concept of objects. Objects can contain data and have actions they can perform.

Many programming languages support object orientated programming to some extent. Major OOP languages include C++, C#, Java, PHP and Python.

The artificial intelligence group at MIT started with concepts of objects in the late 1950s. Later in 1968 ALGOL allow connecting the data structures and procedures. Influenced by earlier languages Smalltalk developers were using the term “object orientated programming” in 1967. By the 80s OOP had become the major programming paradigm used in programming, especially as more languages supported it.

## Classes and Objects

An Object is encapsulated unit of data and behaviour - a type of data structure that has both fields and methods. Fields are also known as attributes, properties or members and hold in formation in variables. Methods are also known as procedures, subroutines or actions define the behaviour of the code with code.

Objects represent both real world objects (such as person or a vehicle) and abstract entities (such as a temperature convertor or a set of co-ordinates).

Objects can contain other objects, known as object composition. For example, an Invoice object could have a Customer object in it. This is a ‘has-a’ relationship as “invoice has a customer associated with it”.

When programming a distinction is made between a Class and an Object. A Class is the code written by a developer while an Object is the in-memory instance of the code within a computer. There is only a single definition of a particular class, but there can be many instances of the same object. Classes are essential template to define an Object.

## Abstraction

## Abstraction refers to the ability to hide complex implementation details and show only the necessary features of an object. This simplifies the interaction with objects, making programming more intuitive and efficient.

The details of the actual implementation become irrelevant if it correctly performs the required task. The programmer does not have to be concerned with how the object calculates its data and has to use it as a tool. You don’t have to know if your bank balanced is stored in a 64-bit integer – you simply know that the BankAccount.GetBalance() method returns “£123.45”.

## Encapsulation

Encapsulation covers the concept of bundling data with the methods used to access and modify it into a single unit. In encapsulation the variables storing the data are hidden and they can only modified by the use of a class method.

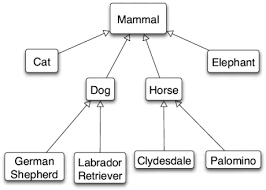
This provides the following benefits:

* Data Protection – by making data private you prevent accidental or intentional modification ensuring data integrity.
* Security – protect sensitive information by restricting access to it.
* Maintainability – changes to the internal implementation are less likely to affect other parts of the code.
* Less complex code – by lowering the connections to other parts of the code, the overall code in a program becomes less complex and less sensitive to change. This lowered coupling between code units reduces the domino effect of an y code changes.

As an example, consider a object for a bank account. Only the BankAccount only has access to the ‘balance’ variable, and it only allows a debit or credit once it has received the correct security confirmation. If this balance was not encapsulated and freely available a rogue or badly written method could incorrectly modify the balance.

## Inheritance

Inheritance is a concept that allow use of properties and behaviours from another class. A child class that inherits from a parent class will have gain all the properties and methods of the parent and can add its own to expand upon the parent functionality. A child can also change the override the parents’ functionality to provides its own version.

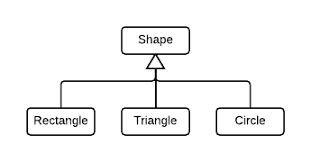
Several classes many have similar functionality – for example Cats, Dogs and Horses each have identical properties eg leg count, heart rate, and several that are unique to the animal such as blood types.

We could implement these methods in each class, each one being identical. Using inheritance, we would define a Mammal class which implements these common properties and then the individual classes can just inherit them and add any additional properties they uniquely need.

## Polymorphism

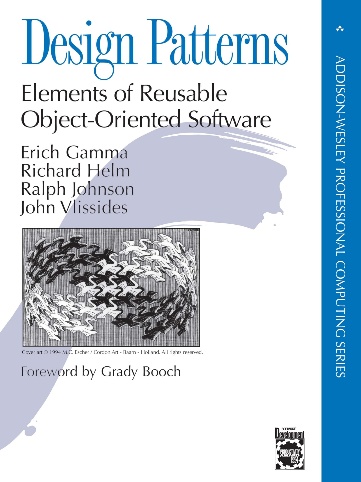
The word "polymorphism" means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Polymorphism allows different classes with different implementations to be accessed through the same interface. In programming, we would define the interface and then write one or more classes that fulfil the needs of the interface. We could then use any object that implements this interface.

 For example, with simple geometric shapes we could have a Circle and Square. Each would have a perimeter length and a surface area. These would look superficially the same, but the calculations would be quite different.

If we want to have a Draw routine, without polymorphism we would need two methods – DrawCircle and DrawSquare. Polymorpsim allows us to define a Shape, and both Circle and Square can inherit from it with a single method DrawShape. The interface for Shape would state that any object inheriting from it would have a perimeter length and a surface area (along with other properties too... FillColor perhaps). Then the class for Circle would only need to define its calculations that are different from other Shapes, the same for the Square.

# Design Patterns

 Design Patterns are pre-used solutions to common problems. They present a methodology that a programmer can use to write a solution to a specific problem knowing that they have used a proven solution.

The book ‘Design Patterns: Elements of Reusable Object-Orientated Software’ published in 1994 began the rise in popularity of design patterns. Written by the “Gang of Four” Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides the book included the classic 23 patterns with examples in C++ and Smalltalk.

A Pattern does not provide any code to solve a problem – only a method. It is still up a programmer to devise the correct implementation.

## Debugging the DPC++ debugger using Visual Studio® Code on Ubuntu ...IDEs

vIntegrated Development Environments (IDEs) are software applications designed to be used by programmers to write code. They integrate features to aide the programmer in the development process into a single environment, otherwise the programmer will have to use several separate applications to perform the same tasks. An IDE can include

* Syntax Highlighting
* IntelliSense and Reference information
* Debugger Tools such as watches and breakpoints
* Compilers
* Unit Testing
* Autocomplete

Some popular IDEs include Visual Studio, Visual Studio Code, JetBrains Rider, Eclipse or Arduino IDE.

# Event Driven Programming

**Event Driven Programming** is a method of programming where the flow of execution of code is responsive to events external to the code. An event is a change in state, or an update, like a button being clicked by a user or a disk inserted into a computer.

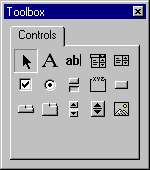
Event Handlers are called in response to a matching registered event and execute the programmer’s code. So, a routine may be registered to respond to a Click event on a particular button. When a user clicks that button, the routine will be executed.

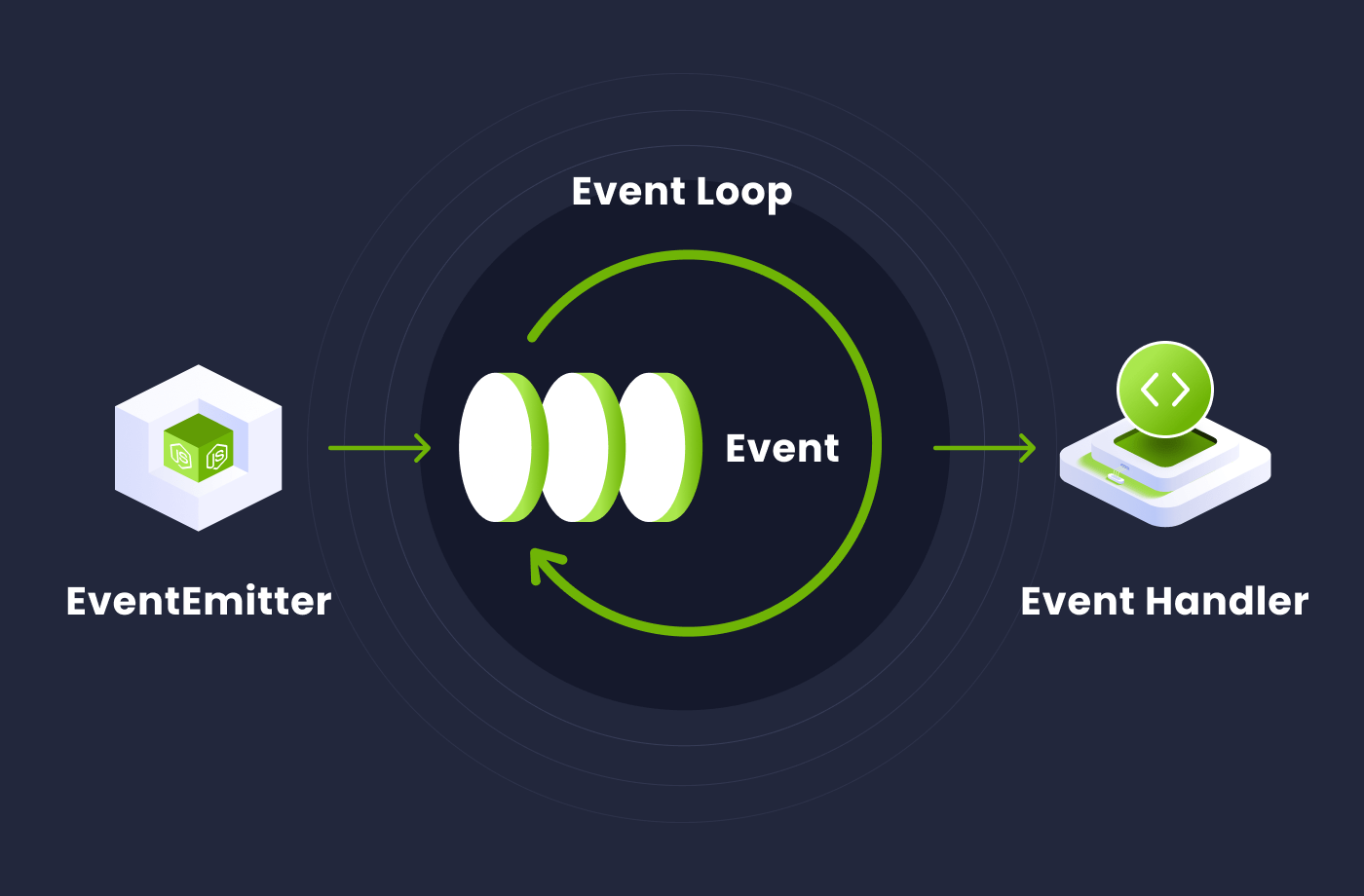
Visual Basic, Visual C++ and JavaScript all provide event driven capabilities.

Different types of events can be generated:

* System Events – e.g. file downloads, hardware signals
* User Actions – e.g. mouse events, keyboard events, touch gestures
* Custom Events – e.g. stock price threshold met, game character loses health

Not matter how the events are generated they are dealt with through the same programming mechanisms.

Visual programming languages such as VB.Net are designed with UI elements such as buttons, menus or radio buttons and these are packaged as **Controls** – a unit which contains the instructions necessary draw it and provide events. The programmer can then write an event handler which can be registered to listen for specific events from the control. With the IDE all the controls can be found in a **Toolbox** – a visual representation of the controls registered with the system.

Event driven programming is built around the **Event Loop** – the event loop is responsible for checking for any generated events and triggering event handlers as appropriate. The event loop processes the events sequentially.

The programmer code is known as an Event Handler. An Event Listener listens for an event during the Event Loop and notifies the Event Handler. Event Handlers can cause problem by stalling the event loop, where the consume system resources for a long time stopping the event loop from processing further events. For example, after a button click the event handler download a file from the internet. While the system is processing this activity not other events are being processed, so the application appears to stall. This can be mitigated by having event handlers release resources so it can process other events, so in this example, a progress bar could be updated or hourglass animations can continue.