Topic 1 Fundamentals of Programming

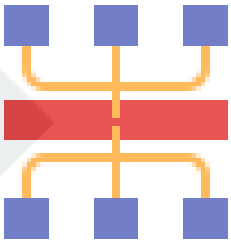
Procedural Programming
Programs which use procedural programming are formed using sequences of instructions which are executed in sequential order, one after another. Subroutines within the program can be called from anywhere within the program. Data is stored in constants and variables, with global data structures being accessible from all parts of the program, and local data structures being accessible from only the area where they were declared.

A Structured Approach to Programming
Using the structured programming approach makes code easy to design and manage. Programs written in this way are designed from the top down, with the most important parts of the program being broken down into smaller tasks which can be solved using a small block of code such as a procedure. These small parts are then joined together to form the whole solution.

Four basic structures make up the structured approach, assignment, sequence, selection and iteration.

Splitting the program into smaller parts allows individual parts to be tested, making it easier to find and rectify any issues. It also makes ongoing maintenance easier and the code easier to understand.

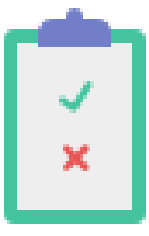
Hierarchy Charts
A hierarchy chart is a picture which shows how the components of a program written using the structured approach fit together. Each part is shown as a rectangle, connected to the other parts used within it. Each rectangle represents a part of the program, and the lines between them show relationships. In complex programs, each rectangle may be linked to more than one other rectangle.



Data Types
Data types are defined by the values they can store and the operations which can be performed on them. Different data types are best suited to different tasks.

- Integer - A whole number, positive or negative, including zero.
- Real / Float - A positive or negative number which can have a fractional part.
- Boolean - A value which is either true or false.
- Character - A single number, letter or symbol.
- String - A collection of characters.
- Data / Time - A way of storing a point in time, many different formats are used.
- Pointer / Reference - A way of storing memory addresses.
- Records - A collection of fields.
- Arrays - An indexed set of elements each of which has the same data type.

User Defined Types
User defined types allow a custom data structure to be created, based around an existing data type. Different programming languages use different methods to create a user defined type.



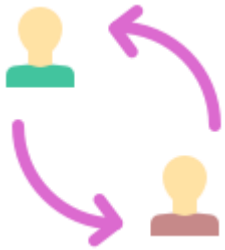
Programming Concepts

- Variable declaration - Creating a variable for the first time, giving it a name and sometimes a data type.
- Constant declaration - The same as variable declaration, but when creating a constant.
- Assignment - Giving a constant or variable a value.
- Iteration - Repeating an instruction.

- Selection - Comparing values and choosing an action based on those values.
- Subroutine - A named block of code containing a set of instructions designed to perform a frequently used operation.

Definite vs Indefinite Iteration

Iteration is repeating a section of code several times.



In definitive iteration, the code is run a set number of times, defined before the loop starts. As an example, this code will output the numbers 0 to 45.

```
FOR Count ← 0 to 45
        OUTPUT Count
        Count = Count + 1
ENDFOR
```

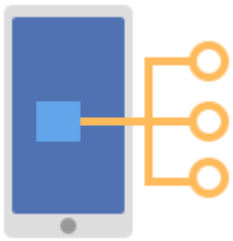
In indefinite iteration, uses one or more conditions to determine how many times the code should be run, the number of iterations is not known before the loop starts. As an example, the code below will get the lighting level whilst the current level is less than 50.

```
WHILE Lighting <50
        Lighting = GetLighting()
ENDWHILE
```

Nested Structures

Selection and iteration statements can be nested, meaning one is placed within another. This might be a loop inside a loop, a loop within a selection, or any other combination. Different levels of indentation should be used to identify the different statements. As an example, the code below shows a while statement within an if statement.

```
IF Waterlevel > 50 THEN
        WHILE Waterlevel > 50
                Waterlevel = GetWaterLevel()
        ENDWHILE
ELSE
        SoundAlarm()
ENDIF
```



Identifier Names

When naming variables, subroutines and other programming constructs it is important to use sensible and meaningful identifier names. This makes it easy for anyone reviewing the code to understand the purpose of the object within the code. Another person with programming experience should be able to work out the purpose of constructs from the name.

Arithmetic Operations

Arithmetic operations are used to work with numbers.

- Addition - Adding together two numbers.
- Subtraction - Taking one number away from another.
- Multiplication - Timesing two numbers together.
- Real / Float Division - Dividing one number by another.
- Integer Division - The same as real / float division, but just the whole number result is given.

- Modulo - Returns the remainder of an integer division.
- Exponentiation - Raising one value to the power of another.
- Rounding - Limiting the degree of accuracy of a number
- Truncation - Removing the decimal part of a number.

Relational Operators

These are used to compare two values

- Equal to 1 =15
- Not equal to 22 <> 76 10 != 20
- Less than 1 < 200
- Greater than 19 > 9
- Less than or equal to 1 <= 27 2 <= 2
- Greater than or equal to 100 >= 24 90 >= 90

Boolean Operators

Boolean values can only be true or false (written as 1 or 0). These operators are used to work with boolean data.

NOT The opposite of a Boolean value NOT 1 = 0

AND Two Boolean values multiplied together 1 AND 1 = 1 0 AND 1 = 0

OR Two Boolean values added together 1 OR 0 = 1 1 OR 1 = 1

XOR True if exactly one of two values is true 1 XOR 1 = 0 1 XOR 0 = 1

Constants and Variables

Both constants and variables allow computer programs to store data. The data stored in a variable can change while the program is running, but the data in a constant cannot change once it has been assigned.

Constants are used to store data which should not be changed, such as the number of hours in a minute, or the value of pi. This allows these values to be given meaningful identifier names, making the code easier for programmers to understand. It also makes it easier to change the value in the future, as this only needs to be done in one place.

Variables are used to store data which does need to change as the program runs. This allows the code to store a value for later use, and a meaningful name to be assigned to the value. Variables are often used with iteration, to record how many times code within a loop has been executed.

String Handling Operations

Strings store one or more characters, often containing an entire word or sentence. The operations below allow computer programs to get information about process the data held in strings.

Length	Returns the number of characters in a string.
Position	Returns the position of a certain character within a string.
Substring	Returns one or more characters from within the string when given a starting position and length.
Concatenation	Joins two or more strings together into a single longer string.
Character to character code	Returns the character code for the given character,
Character code to	Returns the character represented by the given character

character	code.
String to integer	Converting a string to an integer.
String to float	Converting a string to a float.
Integer to string	Converting an integer to a string.
Float to string	Converting a float to a string.
Date / time to string	Converting a date / time data type to a string.
String to date / time	Converting a string to a date / time data type.

Random Number Generation

Most Programming languages can generate random numbers. A function performs a number of mathematical operators on a provided seed value in order to generate a random number. The exact commands and process for generating random numbers varies between programming languages, so you should know how do so in the language you are working with.

Handling Exceptions

The phrase “throwing an exception” means an error has occurred within the computer code. The computer must handle the exception to make sure it does not crash. To do this, it pauses the execution of the code and saves its state before running a special section of code called a catch block. This process stops the code, or the entire computer, from crashing and allows the programmer to provide additional information to the user or carry out certain actions to deal with the error. After the exception has been handles, the program state is restored and execution resumes.

Subroutines

Subroutines are small named blocks of code which perform a specific task. This avoids having to repeat the same code multiple times making the code easier to read and quicker to write. There are two main types of subroutines, functions and procedures. Functions must return a value, whereas procedures do not have to. To access a subroutine, you must “call” it by writing its name in the program code.

Parameters allow data to be passed to the and from a subroutine. They are specified within brackets when calling the subroutine, and contain data needed by the subroutine, for example CalculatePay(Rate,Hours).

Subroutines can return values back to the main program, often the data is stored in a variable. For example, the code WeeklyPay ← CalculatePay(Rate,Hours) calls the CalculatePay subroutine, providing the Rate and Hours variables as parameters, and stores the result in the WeeklyPay variable.

Local and Global Variables

Variables can be either local or global in scope. Global variables can be accessed from any part of the program and exist in memory for the duration of the program’s execution.

Local variables can be accessed only from the subroutine where they are declared, and only take up memory whilst that subroutine is running. This makes them a more memory efficient way of storing data.

Stack Frames and Subroutine Calls

Stack frames store return addresses, parameters and local variables for each subroutine call. Each call is pushed onto the computer's call stack as a stack frame just before the code is executed.

If a subroutine calls another subroutine, this is called nesting. In this case, when the nested subroutine finished, the stack frame is popped from the call stack and this information is used to resume the execution of the previous subroutine.

Recursion

A recursive subroutine is one which contains one or more calls to itself. Recursive subroutines must contain a stopping condition, otherwise the subroutine would never stop. This would cause a stack overflow as more and more stack frames are pushed to the call stack.

Problems which can be solved using recursion, can often be solved with iteration instead, making it easier to code but being less compact.

Object Orientated Programming

Object orientated programs use objects to contain both data and instructions. By contrast, procedural programs use constants and variables alone to store data.

Object orientated programming gives a clear structure making developing and testing easier. It also allows for large projects to be more easily divided between a team. The use of classes allows code to be reused in both the same and other programs

Objects

Objects are created from classes using a process called initialisation. A program can have several different objects based on the same class. The methods and procedures for these objects will be identical, but the actual values and properties will be different.

Classes

A class can be thought as a blueprint or template for objects which specifies the properties (data) and methods (instructions) objects will have.

A class can be written as a class definition listing the properties and methods as shown in the example below. Private methods and properties can only be accessed within an object. Public methods allow an interface for accessing and modifying private properties.

```
Book = Class {
   Private:
      Title: String
      Author: String
      ISBN: Float
      InStock: Boolean
   Public:
      Function GetTitle
      Function GetAuthor
      Function GetISBN
      Function IsInStock
      Procedure Set Details
}
```

Encapsulation

Encapsulation is the process of combining methods and procedures to form an object, an object encapsulates its contents into a single entity.

This allows people developing large programs to be split across different teams and each allocated a class to develop. As long as the team know the methods belonging to other classes, they can develop their own classes without needing to know the specific implementation.

Inheritance

Classes can inherit another class, allowing it to share the properties and methods of another class alongside its own. This is written in the form “is a”, and in the example below Atlas is a Book, as shown by (Book) after the class name. This means that the Atlas class includes all the properties and methods in the Book class as well as its own defined in the Atlas class.

```
Atlas = Class (Book) {
   Private:
      Country: String
      IncludesLandmarks: Boolean
   Public:
      Function GetCountry
      Function GetIncludesLandmarks
      Procedure SetDetails (Override)
}
```

Polymorphism

Polymorphism occurs when objects are processed differently depending on their class. For example, if two objects with the classes Train and Bus both had a ComsumeFuelForOneMile method, it would most likely decrease the Train’s fuel level more than that of the bus.

Overriding

An overridden method has the same name as a method in an inherited class but the methods are implemented differently. In the example above, the word override in brackets after the SetDetails method indicates that it has the same name but is implemented differently.

Association

Associated objects have a “has a” relationship, for example, the Author and Book classes could be associated as Author has a Book. Associated objects form part of their container object as a property.

Aggregation is the weaker association type, associated objects will still exist if it’s containing object is destroyed. Composition is the stronger association type. If the containing object is destroyed the object is also destroyed.

Object Oriented Design Principles

Encapsulate What Varies

Requirements which could change in the future should be encapsulated into a class, allowing changes to be easily made in the future.

Favour Composition over Inheritance

Composition should be used instead of inheritance whenever possible, since composition is a more flexible relationship.

Program to Interfaces not Implementation

This allows unrelated classes to use similar methods. An interface is a collection of abstract procedures which can be implemented by unrelated classes. When a new object is created it can implement an interface which provides it with the correct properties and methods.

Class Diagrams

Class diagrams help to show inheritance relationships between classes.

- Inheritance is shown with unfilled arrows pointing from an inherited class to the class from which it inherits and should always point upwards.
- Association is shown with diamond headed arrows.
    - Aggregation is shown as an unfilled diamond.
    - Composition uses a filled diamond.

If required, the diagram can also show additional information about the class as show below. The top box contains the class name, the middle its properties and the bottom one its methods.

A plus sign means a property or method is public, and minus means it is private. A # symbol means a property or method is protected. Protected properties and methods can only be accessed from the object that declares them.

BOOK
- Name: String - PublicationDate: Integer
+ GetName + GetAge + SetDetails