Programming Techniques

Programming techniques have roughly developed along the following learning curve:

- **Unstructured programming**
- **Functional programming**
- **Modular programming**
- **Object-oriented programming**

Unstructured Programming

Programming is usually learnt by writing small and simple programs, consisting only of one main program. This program contains a sequence of commands or statements that modify data, which is usually global throughout the whole program.

Unstructured programming: Program acts on global data

This technique suffers many disadvantages as the program gets larger, for it will get progressively more difficult to understand, in order to develop and maintain. Also, if a similar statement sequence is needed at different locations within the program, the sequence must be repeated.

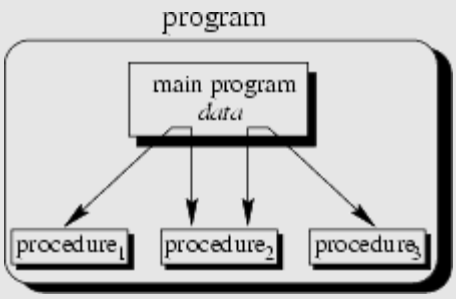
Handling such difficulties in software development has led to the idea to extract similar sequences, name them and offer a technique to call and return from these procedures or functions.

Functional Programming

With functional programming, each sequence of statements to conduct a certain operation is isolated into a single entity, which is called a procedure or function. A function call is used to invoke the sequence of statements of the function. After the function is processed, flow of control returns right after the position where the call was made.

With the introduction of parameter passing, as well as nested function calls within functions, programs could then be written more clearly and reliably. Each function may be checked separately, to verify that it will produce correct results on any given data. Causes of a subsequent error are narrowed down to the functions responsible for related operations.

Functional programming: Program calls functions and passes data



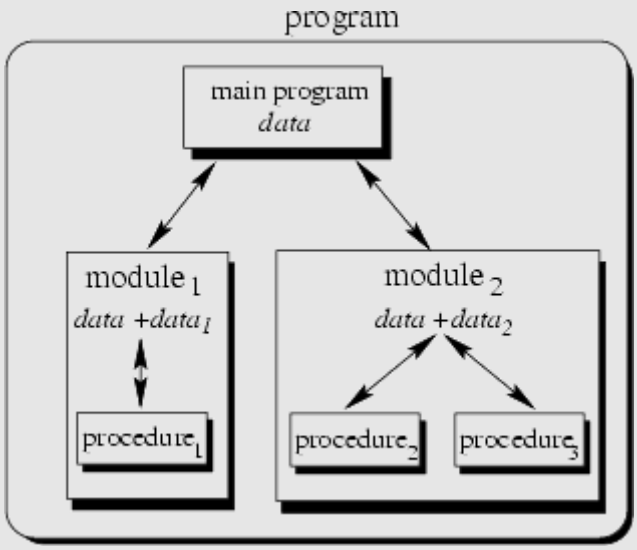
A program can thus be viewed as a sequence of function calls. The main function is where program execution starts and ends. Each function is responsible for passing data to functions called within it, and receiving results from them. The flow of execution may be illustrated as a hierarchical graph or tree.

Hence, a single program uses functions to process the data. To reduce programming effort, usage of general functions or groups of functions in different programs must be supported. Modular programming was developed to allow for grouping of functions into modules.

Modular Programming

In modular programming, functions of some common functionality are grouped together into separate modules. Thus, a program no longer consists of functions only, but is divided into several sections called modules that contain functions. These modules interact through function calls to form the whole program.

Modular programming: Program calls functions in separate modules



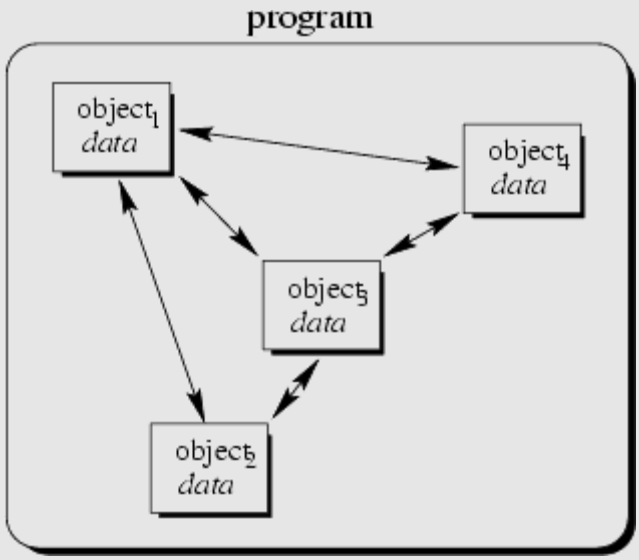
Each module can have its own data. This allows each module to manage an internal state which is modified by calls to functions of this module. However, there is only one state per module and each module exists at most once in the whole program.

There are several problems related to modular programming. A most important one is the decoupling of data and operations, by basing module structure on functionality. In object-orientation, the structure is organized by the data to reduce external data flow. The data representations are chosen first, and the required functionality to operate on the data is added within the same unit.

Object-Oriented Programming

Object-oriented programming solves most of the problems of earlier techniques. In OOP, each program consists of a web of interacting objects, each house-keeping its own state.

OOP: Program objects interact by sending messages to each other



Objects may be seen as modules containing both data and functions needed for operations on that data. Instead of calling functions that must be provided with data, messages are directly sent to objects to invoke their own functions (methods) to operate on their own data (data members). Each object thus keeps its own state.

Data declarations and member function definitions are written once into a type specification (class). Multiple objects of the same class can then be defined by specifying the data member values for each object.

Methods defined for a class are available to all of its objects. When a message is passed to an object to invoke a method, the method uses the data of the particular object.

OOP Features

The main features associated with OOP include:

- **Data hiding**

  Objects usually store, modify, and display their data only via their own methods. This protects the data from unintentional access.

- **Automatic initialization**

  Operations associated with object creation and deletion are executed automatically by using *constructor* and *destructor* methods.

- **Function overloading**

  Functions of the same name may exist, if distinguishable by their arguments. In OOP, classes can have methods of identical names and arguments, because messages address objects of particular classes.

- **Inheritance**

  When a needed class is much like an existing class but requires some different features, the new class is derived from the existing class with minimal extra programming.

- **Polymorphism**

  Objects of derived classes can be processed by the same messages, yet may perform differently as object forms of their parent classes.

OOP Implementation

OOP is especially preferable for the following reasons:

- **Improved reliability**
- **Ease of debugging and development**
- **Increased reusability in other programs**

The most popular environment for OOP implementation has been the C++ language based on C, which is well established as an efficient system programming language. C++ may be regarded as a superset of C with OOP extensions and additional improvements.