Syllabus details of:

3C59

OO Programming course

Aims and syllabus of Theme Modules

Summary of Module 1: In-built Data Types & Simple functions

- **In-built data types**
  - **int**
  - **float**
  - **double**
  - **bool**
  - **char**

....contd

- Simple operations on in-built types
  - arithmetic operators +,-,*,/,=,+=,-=,.. etc..
  - in-built functions cos, sin .. log,..etc ..

- Packaging operations inside simple functions
  - declaration of function name
  - declaration of arguments
  - return type
  - body of the function

- "Hands on" basics of how to make a program run
  - compile link and run
  - include other files containing functions you need

Aims of this module

In this module we introduce the in-built types (integer, floating point, char and bool) with which most students will already be familiar.

We emphasise that it is important to declare the types and names of variable needed in your piece of computer code.

We then introduce the basic operations one can do with in-built data types. This includes arithmetic operations as well as some in-built functions.

Finally we show how repetitive or tedious bits of code can be packaged inside simple user written functions.

Summary of Module 2: User defined Data Types & Operations upon them

- **Inadequacies of in-built types**
  - **limited use**
  - **problems generally contain abstract entities**
  - **difficulty in referring to a single instance**
  - **difficulty in passing a group of variables as arguments**

- **User defined data types**
  - **classes**
  - **declaration of instances of a class (object)**

- Operations on object member variables:
  - access to members of an object
  - simple manipulations of members

- Operations on objects via methods
  - Use of "Methods" to hide the implementation of an object from a user

- First steps toward data encapsulation
  - Consolidation of the ideas of:
    - Object consisting of "state" and "Methods" to use/change it
    - "private Members", I.e. hiding data members from user
    - "public Methods", I.e the things a user program may use

Aims of the module

In this module we show that the simple in-built data types are not generally convenient for representing the types of entity you find in even a simple programming situation. In general you need several integers, floats, characters ...etc.. to do this, and if used in raw form these would lead to excessively messy and cumbersome code.

This leads us to introduce the idea of user defined data types which contain variables for all the attributes of an entity. In C++ this means the use of **`classes`**

We introduce the word **"object"** to mean **"instance of a class"**

We then show you how you can manipulate the member variables of such objects.

However we then convince you that actually manipulating the member variables of such objects is bad news. Not only is it still cumbersome, it is fundamentally dangerous as it means the innerds of the class can then never be changed without upsetting users.

To avoid this we introduce a set of functions to separate the user from the implementer. We call these "Methods of the class". These Methods are the only things allowed to access the variables in the class.

Finally we consolidate the ideas we have introduced by taking an abstract look at Data Encapsulation

Summary of Module 3: Algorithm steering elements

- **The if statement**
  - **if { }**
  - **else if { }**
  - **else{ }**
- **The switch statement**
  - **enumerated constants**
  - **switch / case**

- The for staatment

  for(... ; ... ; ...) { }

The do and while statements

  while(test) { }

  do { } while(test)

Aims of the module

The previous modules have concentrated on programming philosophy. As such they have until now only touched upon very basic C++ language components.

In this module we extend this to cover C components which are in practice needed in almost all programming situations. These are the constructs needed to make branching decisions in programs.

Summary of Module 4: Handling collections of objects: A simple introduction to the vector class

- The need fo collections of things

- The vector class
  - making an collection using vector<SomeType>
  - making an collection of a known size
  - adding items using `push_back( )`
  - finding the number of elements using `size( )`
  - accessing/changing elements using [ ]

Aims of this module

There are very few areas of scientific coding where you do not immediately run in to the need to store and manipulate a collection of some object. This might be a set of data readings or a collection of ThreeVectors representing tracks in a detector ...etc.

In the abstract examples you might need a collection of BankAccounts administered by one bank, or a collection of books in a library.

In this module we introduce you to the simplest of a set of standard library classes supplied with C++, the vector class.

This is designed to add and remove elements to in an easy way, and has methods to let you manipulate the collection.

This module will also be your first example of use of an object where you ONLY know about its methods, but you neither know nor care how it works internally.

Summary of Module 5: I/O

- **Input and output**
  - **cout << tosend to keyboard**
  - **cin << to get from keyboard**
  - **File i/o using streams.**

Aims of this module

You do not get far with any program without the need to either inputto and/or output something from the program. This is particularly true in scientific analysis where the very minimum is likely to be a file of data points to analyse.

This module covers the things you need to know to deal with user and file I/O

In several precediing modules we have used "hands on examples" cout<< and cin>> to write things to the screen and read in from the keyboard. In this module the aim is to cover these more formally.

We then show you how to get things from files.

Finally, and a little out of place, we introduce the string class which provides a convenient way to deal with text strings. We do this here as it naturally fits with I/O.

Summary of Module 7: Constructors and Destructors

- **Constructors**
  - **Constructors are called automatically when you make an object**
  - **You can provide several constructors with different argument lists**
  - **The correct constructor is called according to the arguments to supply when making the object**
  - **Copy constructor**
- **Destructor**
  - **The destructor is called when an object get destroyed. Its function is to cleanly end the life of an object**

Aims of this module

We have previously introduced the concept of a “class” as a construct which combines member variables and methods in a formal way.

In this module you will learn about a new and vitally importantpart of a class which determines how an object gets made.

This all revolves around the idea of "Constructors" which are special methods which get called automatically to initialise objects.

Associated with these are "Destructors" which get called when an object goes out of scope or is deleted.

These are fundamental new concepts in OO and you must use these from early on. Therefore this module covers their use in some detail.

[As a part of this we have to cover "references"]

More on methods:

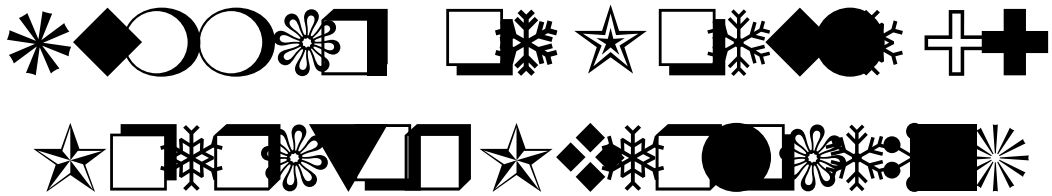
- Overloading of methods
  - You can have many different methods with the same name provided they have different argument lists
- Private methods and self messaging
  - If you want to write a method purely for internal use by the class, then you should make it a private method
  - Methods of a class can call other methods of the class.
- const methods
  - If a method does not change the state of an object then you should always make it const
- const arguments
  - If a method does not change an argument then make the argument const

Aims of this module

We have previously introduced the concept of a "class" as a construct which combines member variables and methods in a formal way.

So far, however, we have only looked at very simple aspects of methods of a class. In particular we have only written the methods using simple features (i.e those which you might be familiar with from simple C or Fortran)

In this module you will learn about some features of C++ which allow you to use methods of objects in a much more flexible and controlled way.



- Overloading +
  - equivalence of "+" with "operator+"
  - declaring and writing the operator+ method
  - return values from operator+

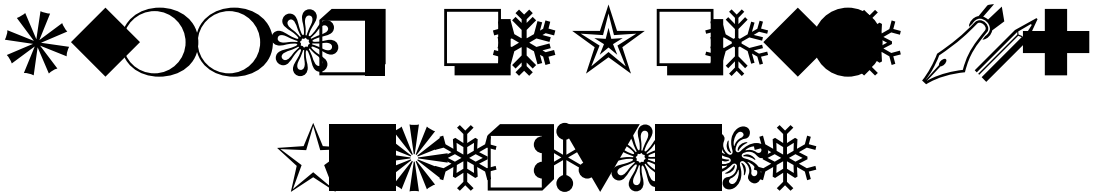
- Overloading of =
  - return value from operator=
  - use of *this
- Overloading of ==

- All the others needed for the Complex class

Aims of this module

In this module we introduce the feature of C++ which allows you to custom define the action of operators for classes.

These include =, *, +, +=,....etc
This feature gets used for several simplifications which you need to know about.



- Basics of inheritance
  - class child : public parent
  - adding of methods
  - adding new member variables
  - use of inherited methods
- Over-riding base class functionality
  - over-riding a method of the base class
  - calling the base class method explicitly with the scope resolution operator
- Rules for constructor inherance
  - default is that base class default constructor gets called
  - for any other action you must explicitly call the base class constructor you want

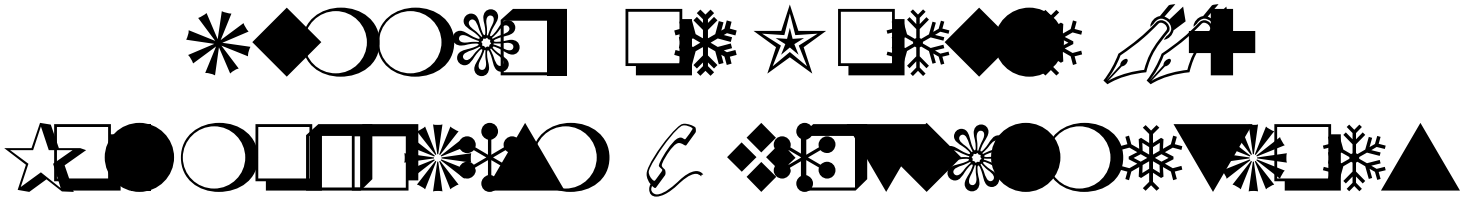
Aims of this module

The aim of this module is to familiarise students with the most simple aspects of inheritance

Inheritance allows you to define a class based upon a pre-existing class.

The new class is said to "inherit" all of the public services of the base class. In other words you get all of the methods of the base class free. Inheritance is a key component of OO languages and by the end of this module it ios intended that you will be able to write a simple inherited class.

The module also covers the related topic of overriding base class methods.



- **Polymorphism**

  Use of sub-classes polymorphically via references or pointers to a base class

  Use of virtual methods and the virtual keyword

Aims of this module

A key feature of object oriented programming is the ability to use objects "polymorphically"
This is one of the most powerful featrues, and you cannot be said to be truly writing OO code unless you understand and can use polymorphism.

Polymorphism allows code to be written which operates upon a base class type, but which will also immediately (without modification) work on any other concrete classes which inherit from this base class, and which may be invented at any time in the future.

We explain this featureand describe a context in which it might apply.

Technical Modules

Summary of Module : Pointers

- **Pointers to primitive in-build types**
  - *** to de-reference a pointer**
  - **& to take the address of**
- **Pointers to objects**
  - **The same as for in-built types**
  - **In addition the -> operator for invoking methods**

- Use of Pointers to pass arguments
  - A (not recommended) alternatice to references
  - Historical as pointers existed before references

- Use of new and delete
  - Dynamic allocation of memory
  - Returns a pointer to the allocated memory
  - You are responsible for keeping hold of the pointer
  - you must delete the allocation when finished.

- Important use for destructors
  - Destructors should delete any memory which an object allocates with new UNLESS responsibility for the pointer has been passed to some other object.
- this->