Topic 9 – Object Oriented Programming

The Object-Oriented Paradigm

Introduction

- In Topic 5, we looked at top-down design and structured, modular programming.

- Object-oriented programming (OOP) is an entirely different way of approaching the process of writing a program.
  - In fact, in its most complete form it heavily influences both the design process and the final implementation.

- As we will see in this topic it also determines the structure of the way the solution to a problem is divided up in a very fundamental way.

- Object-oriented programming can be done in the C language.
- However, the language was not designed for this and involves use of some fairly advanced features.
- Therefore, in this unit, we will only be looking at the theory of object-oriented programming.

The Structured Model

- In a procedural program we have separate modules of code (functions in C) which do various things.

- We also have data which is declared inside modules and can then be passed around between modules.
- However, the data doesn't actually belong to any of these functions, although the variables that store it do.
- Instead the data is “just passing through”.

- As we end up with more and more complex data structures to model the problems we are trying to solve, there is more and more work maintaining and organising these structures.
- Therefore, our program will include a large number of functions that need to be called in order to ensure these data structures work correctly with the program.

Programs are Made of Objects

- Procedural structured programming that is used in languages like C involves taking the high-level algorithm which solves a problem and dividing it up into the individual steps.
- Each of these steps becomes a separate module in our program.
- Often these steps are broken up further into substeps which means that a function will call other functions in order to complete its task.

- With the OO paradigm, the program is divided up into objects.
  - These sometimes relate to physical objects, e.g., a program managing data about people might have a *Person* object.
  - Of course they don't have to be physical things but by making an object for them they can be treated a little like they are.

- One major difference here is that OO is much more data-centric in terms of developing a solution whereas the traditional approach is much more concerned with the steps involved in solving the problem (code-centric).

- Of course, OO programs also contain code to do things with the objects otherwise they couldn't do anything.

- But this is all very vague so what exactly is an *object*???

Object = Code + Data?

- In OOP the objects are made up of both code (things that the object can do) and data (information about that particular object).
- In this way OOP bundles together both the operations that relate to the object and the information that has to be stored in order to perform those operations.

- This doesn't happen in procedural structured programming because the actual data doesn't really belong anywhere.

- So our *Person* object might store data like the person's name, their phone number and their email address.
  - This makes it quite a lot like a `struct` in C.

- However, unlike a `struct` the code to process that data is bundled together with it.
- So there might be methods in an object to set the person's name to a given value and/or to send an email to them.
  - So an object is a little like a `struct` that also has functions (methods) as part of it to perform operations on the data it stores.
  - These operations depend on what is appropriate to the object and what the program needs to do.

Classes vs Objects

- So an object is made up of code and data and, in OOP, programs are made up of objects.
  - However, object-oriented code is often said to be made up of *classes*.

- The difference between classes and objects is actually a little subtle but is crucial to understand.

- If we have a *Person* object then that stores data about a specific person and can perform various operations on the data relating to that person.
- However, there may be lots of different people for which data has to be stored so separate objects will be needed for each.

- A class therefore is a general specification of an object, a little like a blueprint or a design.

- We can write a *Person* class and this won't actually store data about any particular person but rather defines what data is stored and also what operations can be performed on this.
- We can then create any number of *Person* **objects** based on this *Person* **class**.
  - This is a little bit like the data type `int` and declaring a specific variable of type `int`.
- We say that an object is an *instance* of a particular class.

- Classes are therefore effectively complex data types and objects are like variables declared from that type.
- The difference between variables and objects though is that objects contain code as well as data and can perform operations on the data they hold.

Modular Programming Principles and OOP

- In Topic 5 we looked at a number of principles relating to modular programming, namely abstraction, code re-use, high cohesion and low coupling.

- In general these principles apply very strongly to OOP as well.

- Most OO languages provide a number of features to enhance and enforce *abstraction*.
  - These mean that people who are using classes do not need to know the details of how those classes work, just what they do.
  - In OOP this is commonly called *encapsulation*.

  - In fact, OOP enforces abstraction even more rigorously than non-OO programming.
  - Access to either the code or data inside of an object is often restricted.
  - In particular, data inside of an object is rarely accessible outside of that object.
  - These are referred to as *private data members*.
  - However, code modules usually are accessible.
  - Therefore, access to the data inside of an object must instead occur through the function.

  - The designer of the class specifies which pieces of data/modules are accessible, and which aren't.
  - This allows control over how the program at using the class can interact with it, and thereby help ensure it is used correctly.

- *Code-reuse* is a critical part of OOP in that the classes created must be general “blueprints”.
  - Objects can then be created as instances of these classes facilitating code re-use.
  - In more advanced OOP you can actually take an existing class and extend it by adding new features and adapting it to your specific requirements without needing to start from scratch.
  - This is called *inheritance* and allows even more code re-use.
  - Some OO languages can also support the ability to apply a module to different combinations of types of data given as parameters.
  - This is called polymorphism and can also assist in reuse.

- *High cohesion* applies to both the design of the classes and their code modules/methods.
  - Classes only store data that is relevant to their task.
  - Methods are often particularly narrowly focused in terms of the tasks they perform.

- *Low coupling* also applies to OOP.
  - In some ways it is slightly less significant since less data is passed around between modules.
  - However, in others it is more important as data is generally completely inaccessible outside of the object.

BUILDING CLASSES

Methods and Constructors

- So to do OOP we need to first define what classes are needed.
- And classes are made up of code and data.

- Code modules in object-oriented languages are generally called *methods* (similar to *functions* in C).
- So the code that makes up a class are all part of methods.

- The methods found in classes perform various operations that relate to what the class is for
  - Specifically they perform operations on the data that is part of the class.

- However, there is also a general category of methods that are involved in setting up each instance of the class (object) for use.
- Whenever a new instance of a class is created, one of these methods is called to make it ready for use.

- This special type of methods is called a *constructor*.

- Constructors look just like ordinary methods except they have the **same name** as the class which they are in.
  - Note that since there can be a number of different ways to initialise new objects, there is often more than one constructor for each class.

- Creating a new object involves calling the constructor, which sets up all of the data inside of the new object and reduces the work that the programmer needs to do.

Access methods

- As indicated previously, the data inside of an object is generally not accessible outside of that object.
- To access these private data members, instead the programmer using the class must call a method.
- These methods are known as *access methods*.

- There are two basic categories of access method.

- *Set* methods allow the programmer to change the value within the object.

- *Get* methods allow the programmer to retrieve a value from within the object.
    - This could be a value that has been calculated by the method
    - Or it could simply be the value of the private data member, returned by the method.
    - The programmer won't necessarily know which applies, which enforces encapsulation and abstraction.

- Using access methods allows the designer of the class to control how the programmer interacts with that class.
- For example, the programmer cannot change the values of private data members, except through an access method.
- The access method can then impose restrictions on what can be done.
- For example, a Get method might refuse to give the programmer a value if the variables concerned don't yet have values in them.
- Similarly, a Set method might prevent the setting of invalid values.

BASICS OF OO DESIGN

- Writing a program in a structured, procedural language like C largely involves working out what steps you need to perform.
- Applying a process like top-down design, simply breaks those steps up into a hierarchy.
- Therefore, structured design focuses on the *code* and what functions need to exist to solve a problem.

- Writing a program in an OO language is quite different.
- The design process involves looking at the *data* involved in the problem, and then working out what needs to be done in processing that data.
- Therefore, OO design is very data-centric.

- Writing a program in an OO language therefore generally involves:
  - Defining all the classes that are needed to solve the problem.
  - Creating instances of those classes (objects) that are needed.
  - Writing code to execute the code (called methods) that are part of these objects.

- Think about a simple program like the triangle program from the assignment:
  - What data would this programme need to store?
  - What processing needs to be done on that data?
  - What other methods might be needed?

SUMMARY

- This has just been a very basic introduction to the concepts involved in object-oriented programming.
- If you go on to do more computer science, you will study object-oriented languages and OO design.

- You would not have to apply OO concepts in any of the programming for this unit.
- However, remember that the theory of these concepts is examinable!