

## COMP3062    Advanced Databases

### 99-04    Assignment 1

## 1 General Information

The first assignment is a C++ programming assignment, and is due at 1700 (ie: 5pm) on Friday of week 8 (10th September). It is worth 20% of the total assessment.

### Note

- Your program must be written in C++, and compile under the Sun Microsystems CC compiler as installed on *iwaki* (ie: don't use g++, gcc, or cc).
- You are responsible for your own work. The departmental policy on plagiarism is available in the departmental student handbook, and will be enforced.

Your task is to write a program called `unit-db`, which is an in-memory database application. It should “populate” its database from data files, and then respond to queries found in a file whose name is a command line argument. Further details can be found in the subsequent sections of this document.

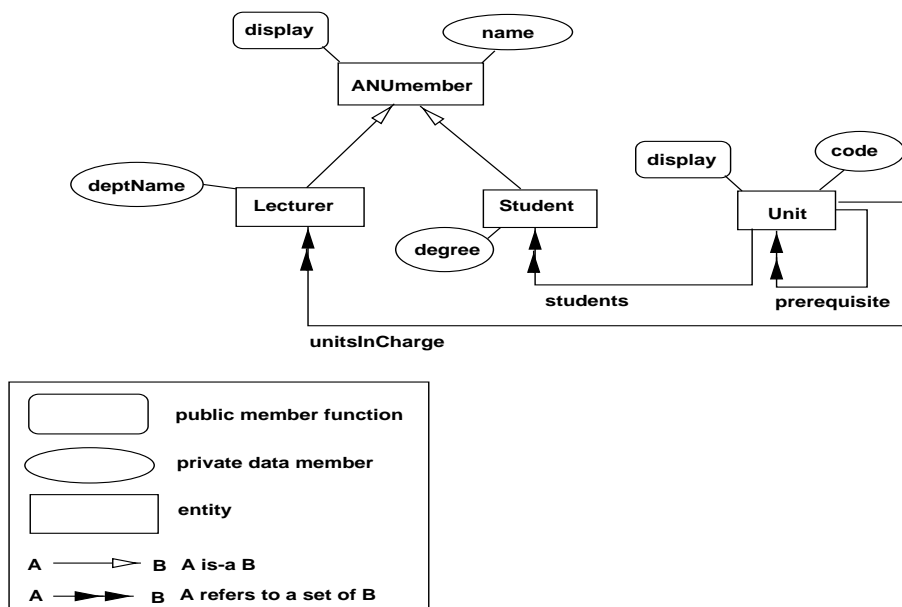


Figure 1: A Database Schema

## 2 Database Schema

A database schema is given in Figure 1.

- **Entities:** All entities given in Figure 1 have to be implemented with C++ classes. The is-a type hierarchy should remain unchanged.
- **Data Members:** All data members must be private data members. As for a data member, called  $AB$ , associated with  $A \rightarrow B$ , you have to implement it as a pointer pointing to a class, called  $BSet$ , which is a class to manage a set of  $B$  objects. The details will be given later.
- **Constructors and initial values:** You must provide constructor(s) for a class to initialise ALL data members defined in that class and its super classes.  
For simplicity, the default `degree` of students is `BSEng`, and the default `deptName` of lecturers is `DCS`.
- **Member functions:** All the functions given in Figure 1 must be public member functions. The `display` function defined in `ANUmember` must be a virtual public function which will be inherited to its subclasses. The `display` function must print all data members once and only once.
  - On `ANUmember`: name only.
  - On `Student`: name and degree.
  - On `Lecturer`: name and `deptName`.
  - On `Unit`: code, a set of student names, and a set of lecturer names.

**Note:** You can add any number of member functions onto a class but you cannot add/remove data members.

- **Sets:** In order to handle a set of objects of a certain class, you must define two extra classes. One is a “set” class and the other is an “iterator” class. For example, for `ANUmember` class, you have to define an `ANUmemberSet` class and an `ANUmemberIterator` class. The former is used to maintain a set of `ANUmember` objects. The latter is used to access every `ANUmember` object in an `ANUmemberSet` object. The same applies to all the classes. An example is given at `/dept/dcs/comp3062/public/ass1/src/iterator` which shows how to use these two classes together to manage a set of objects.

When you need to reference a set of objects in a class, for example, to reference a set of students through the data member `students` in the `Unit` class, you have to define the `students` as a pointer to a `StudentSet` object. A `StudentIterator` must be used to access all student objects in that set object.

- **Extensions:** An extension of a class,  $A$ , is the set of ALL objects in the database belonging to class  $A$  and any subclasses of class  $A$ . In other words, if class  $A$  is a superclass of class  $B$ , the extension of  $A$  includes the extension of  $B$ . For example, a student object should exist in both the extension of `Student` and the extension of `ANUmember`.

In order to simulate a database system, you have to use the following global variables in your program.

```
ANUmemberSet *ANUmemberExt;  
StudentSet   *anuStudentExt;  
LecturerSet  *anuLecturerExt;  
UnitSet      *unitExt;
```

The reason that we need to keep these extensions is that we need to access, for example, all students in our database.

### 3 Coding Style

Each class must be implemented in three files. As for ANUmember class, you need `ANUmember.h` to keep the definitions including ANUmember class and the other two classes for “set” and “iterator”. You need to have `ANUmember.cc` to implement member functions for the ANUmember class. You also need to have a file called `ANUmemberSet.cc` to implement the “set” and “iterator”. Use the files in `/dept/dcs/comp3062/public/ass1/src/iterator` as an example.

### 4 Data

These sorted files are given at `/dept/dcs/comp3062/public/ass1/data`

- The `student-unit` file contains a list of pairs. Each pair represents a relationship between a student name and a unit code the student takes.
- The `lecturer-unit` file contains a list of pairs. Each pair represents a relationship between an employee name and a unit code.
- The `prerequisite` file contains a list of pairs. Each pair represents a relationship between two units. The second is a prerequisite of the first.

Your program should open these files *in situ*, ie: don't copy them to, or expect them to be in, the current working directory.

An example is given showing how to read data from a file  
`/dept/dcs/comp3062/public/ass1/src/io`

### 5 Construction of a Database in the Memory

You need to construct an in-memory database based on the database schema using C++. An in-memory database is a set of extensions. You need to insert objects into the corresponding extensions.

### 6 Queries

You need to support the following queries.

1. Given a string which can be either `ANU-Ext`, or `Student-Ext`, or `Lecturer-Ext`. Print the result of the `display` function for all objects in the corresponding extension.
2. Given a unit code (a string). Print the result of the `display` function for the corresponding unit object that has the same unit code in the `unitExt`.
3. Given the full name of a Lecturer, print the list of units he/she is involved with.
4. Given two full students names. Print the unit codes both students are doing. Note: students names might not be unique.
5. Given two unit codes. Find out if the second unit is a prerequisite of the first unit (directly or indirectly).
6. Given a unit code, list all units students have to do in order to do that unit (directly or indirectly).

The command `unit-db` shall take a file-name as its argument.

`unit-db file`

In the `file`, each line is a pair of the question number given above and the query. A question number can appear multiple times in the file. The number of the lines in the `file` can vary. Examples are given below.

```
1 ANU-Ext
2 comp2031
3 Mr.Right
2 comp2030
4 Mr.A Mr.B
5 comp2030 comp2029
6 comp2033
0
```

where 0 indicates that there are no more queries, and that the program should exit normally.

## 7 Submission

Use the on-line `mark` tool to submit a `Makefile` and all necessary source files (`*.cc`, `*.h`). Running the command

```
make unit-db
```

must produce an executable called `unit-db`.

All files must be submitted by 1700 (ie: 5pm) on Friday of week 8 (10th September).