Introduction to Practical Exercises

Overview

The exercises in this guide are intended to consolidate the material covered in the course while your memory is fresh and while there is an instructor on hand to ask!

Each section describes overall objectives, reference material, an overview of the exercises and the questions themselves, including an optional section at the end.

This course will either be running on Microsoft Windows or on Linux. Mostly the material applies to both operating systems, but where there are differences they will be clearly marked.

Software Set Up

First determine which operating system we are using! If the course is running on a QA site then you will be running a version of Windows with an optional Linux Virtual Machine. Your instructor will show you how the start the VM if you opt to use Linux - its use is optional.

Both operating systems use a similar directory (folder) structure. **On Microsoft Windows** these are held on the C: 'drive', and **on Linux** in your home directory.

The top level directory is named **qacadv**, with a sub-directory for each chapter for which there are exercises. Within these sub-directories are source file templates for you to edit and complete, files for building your application (see below) and sometimes data files for testing. Should you inadvertently make a mess of these (it happens to the best of us) then a backup is held under sub-directory **Original**, which you can copy. Suggested solution code is held under sub-directory Solution. You are probably aware that to any problem in programming there are many solutions, so do not worry if your solution is different to ours – it might be better!

Windows

On Microsoft Windows the tool provided is Microsoft Visual Studio. A summary of its use is given in an Appendix, and your instructor will guide you through the basics. Remember, this is not a course on using Visual Studio!

Programs within Visual Studio are maintained within **projects**, and these in turn are held within **solutions**. Complex arrangements can be built, but we shall keep it simple and (mostly) arrange one program to one project with each project in its own workspace. When starting a new exercise we will tell you to open a new workspace, which should load the source code for you. Sample solutions also have their own **solutions** (suffixed **.sln**) in the **Solutions** sub-directory.

Linux

On Linux you have a choice of editors available, and the compiler in use is the GNU C compiler. An introduction to Linux and suggestions for editors is given in an Appendix, and your instructor will guide you through the basics. Remember, this is not a course on using Linux! If you are not familiar with UNIX/Linux command-line basics then you are probably better using Windows.

On Linux the source files are regular text files, with .c or .h suffix. The suffixes are not meaningful to the operating system, but are to the compiler. Having made your changes you can compile and link your program by calling the compiler yourself, or you can use the 'make' files provided. The later is the simplest, just make sure your current working directory is correct, then type **make**. This will attempt to compile and link all the programs in the exercise, to build an individual program type **make** followed by the program name. To run the program, type its name on the command line. For example, to compile and run myprog.c:

```
make myprog
     myprog.
```

Help and Hints

Some of the questions invite you write a complete C program from scratch, but in some cases we provide a simple code template to get you going. These code templates relieve you of some of the mundane coding, so that you can concentrate on the relevant (and more interesting!) aspects of the program.