CAAM 420 Fall 2014 Overview *

T. Warburton

August 25, 2014

1 Course Description

The goal of this course is to make the student familiar with a set of basic tools for solving problems in scientific computation and documenting the solutions: high-level programming languages (C, C++, Fortran 77), program development and maintenance tools (editors, make, shell programming, debugging, source control systems), and some documentation and and typesetting utilities. The guiding principle of the course is this: if someone else has written a competent solution of your problem, use it! In particular, most of the numerical tasks we undertake will be solved using library code downloaded from the Internet; we will concentrate on writing the connective tissue in various ways to make a living application with effective input and output.

C is the first high-level programming language discussed in this course - it will occupy roughly the first half of the semester. C is widely used in both scientific and commercial programming. It's inextricably linked to the Unix operating system and its various descendants - in fact the inventors of C (Ken Thompson, Denis Ritchie, Brian Kernighan, and others) were also the inventors of Unix and wrote C in part to create a portable alternative to assembly language in which operating systems like Unix could be expressed. We will work exclusively in a Unix-like environment in this course - you'll most likely work in Linux, an open software replacement for the proprietary Unix-related operating systems that predominated in the 1980's and 90's. Mac OS-X has another Unix clone at its heart, and is a perfectly satisfactory environment for the sort of programming we will do. A great deal of useful numerical software is written in C or accessible to programs written in C. Moreover, the essential concepts embodied in C are ubiquitous in scientific computing (in fact in programming generally).

1.1 First Half: the C programming language

The first half of the course will cover these aspects of C:

- Basic types and operations on them.
- Control structures.
- IO and formatting.
- Arrays and pointers.
- Memory allocation for arrays (static and dynamic).
- Structures, typedef.

The main course reference for this material is the book by the inventors of C, Kernighan and Ritchie's The C Programming Language, 2nd edition. This book ("K&R") is more a reference book than a text: while the first 185 pages have narrative form, it's really hard to simply sit down and read. On the other hand, you will find yourself returning to it whenever you program in C. It is truly the fundamental reference for this topic. I have added a couple of additional recommended texts on C and related topics to the reading list, and will add more as the semester wears on. Many students find the text by Deitel & Deitel helpful; it is

*Adapted from the CAAM 420 Fall 2010 Notes of **Professor Symes**

long and discursive, like a textbook, whereas K&R is short and almost too concise (like Unix). I find Peter van der Linden's book illuminating on some topics that K&R leave obscure.

As the examples become more complex, you will begin to appreciate the need for systematic approaches to software organization and maintenance, source code management, and program documentation. This course will introduce basic methods and tools that address these issues:

- `emacs`, LaTeX
- `make`, `gdb`, `valgrind`
- Libraries
- Subversion

Much high-quality numerical software was written in Fortran 77 during the 80's and 90's, and this code still has considerable value. Fortran 77 is the final form of one branch of the Fortran language, the first high-level (portable) computer language designed to support scientific computing (a few comments on Fortran 90 and descendents, another branch of this language, are offered below). It was introduced in the 1950's, and is still widely used in science and engineering. The branch represented by the Fortran 77 standard is essentially a subset of C with different syntax. With a bit of additional information regarding linking, this large body of useful code is accessible to programs written primarily in C. You will build several applications (linear system and differential equation solvers) by calling Fortran library code from C main programs.

1.2 Second Half: the C++ programming language

The second part of the course is an introduction to C++. C++ is a superset of C, and offers a number of enhancements of C's basic features: for example its dynamic memory management facilities are superior replacements for those of C. However C++ is much more than "a better C". It solves a problem that will become apparent when we try to write (for example) a truly general linear system solver in C: our attempts will become entangled in the (irrelevant) particulars of data representation. Solvers written for simple C arrays will not work for multidimensional C arrays, nor for data arrays stored on disk (out-of-core). Yet all of these storage modalities are merely implementation details, which do not appear in the solver's mathematical description.

The missing ingredient is ability to define new types and their behaviour, such as "vector" and "finite element mesh", which can be manipulated regardless of the details of their implementation. Programming centered around the logic of types has come to be called object-oriented. Languages supporting object orientation offer the programmer facilities to define new types and the relations between them, in particular abstract types which encompass more than one possible implementation (eg. "finite element mesh"). Proper type definition permits clean separation of levels of abstraction within a programming project, thus leading to code that is easier to produce, use, and maintain. Object orientation has been the dominant paradigm in commercial programming for 20 years or more - arguably, software innovation would have choked on unnecessary complexity without the introduction of object-oriented techniques. These ideas have had less impact in the scientific programming world - so far.

C++ is one of the two widely used programming languages offering support for object-orientation, the other being Java. Other languages with object-oriented features include Objective C, C#, Fortran 2003, Eiffel, and Python. We choose to work with C++, partly because it is an extension of C, hence permits access to the reasonably high level of performance obtainable with C, and partly because it supports two styles of abstraction (inheritance-based and generic), both of which are quite useful. The agenda for this part of the course is:

- Classes - data and procedure encapsulations in user defined types
- Memory management, constructors and destructors, access control
- Inheritance and virtual functions
- Templates and the standard library

- As for C, I choose to recommend the primary reference book as our text: The C++ Programming Language by Bjorn Stroustrup, the inventor of C++. Like K&R, Stroustrup's book is not an easy read, but it is the ultimate reference on its topic, and you will return to it so long as you program in C++. Other supporting references are provided in the reading list (and more will be added as the semester progresses).

In Fall 2012 we covered additional material on using the open graphics library (OpenGL) for rendering 3D objects, the graphics library utilities (GLUT) and the open audio library (OpenAL) for rendering stereo sounds.

1.3 Third Half: other languages

Two topics not covered deserve a couple of words. Fortran 90 (and its minor upgrade, Fortran 95) is a major extension of Fortran 77, quite popular in some engineering disciplines. I have not elected to include it in this course for two reasons: (1) most high-quality Fortran library code is still in Fortran 77, not Fortran 90; (2) it is a large language, and would take up considerable time beyond what is needed for Fortran 77. Its principal advantage over Fortran 77 is its Matlab-like array syntax, which however hides some very serious performance issues. Its data abstraction facilities are no more extensive than C's; it does not support OOP in any meaningful sense.

The newer Fortran 2003 standard does define support for both inheritance-based and generic polymorphism, but the compiler situation is murky at the moment (2010) - and, once again, the newly introduced features are roughly those which have been available in C++ for 15 years. Java was a harder call. At the moment, C++ is easier to adapt to scientific applications with reasonable levels of performance. Despite the messiness of C++, it therefore gets the nod for (perhaps transient) utility reasons.

For a detailed description of the topic sequence, see the Course Syllabus with its links to weekly lecture note sets.

1.4 Objectives

Course Objectives: CAAM 420 students learn to devise and implement concepts, algorithms and methods derived from applied mathematics using the C and C++ programming languages.

1.5 Course Outcomes

After completing this course, students should be able to:

1. Navigate comfortably within the Linux operating system;
2. Manage a computing project with the Subversion source management system;
3. Create C/C++ computer programs for mathematical applications;
4. Use the GNU make system (i.e. create and use Makefiles);
5. Debug C/C++ programs using the GNU debugger gdb;
6. Find memory errors in C/C++ programs using Valgrind;
7. Evaluate bottlenecks in C/C++ programs and take steps to optimize code performance;
8. Validate and verify C/C++ programs;
9. Typeset an *original* academic paper in LaTeX;
10. Document a coding project;
11. Collaborate on a coding project;
12. Use mathematical libraries (BLAS, LAPACK, FFTW, UMFPACK);

1.6 Notice

Any student with a disability requiring accommodation in this course is encouraged to contact the instructor during the first week of class, and also to contact Disability Support Services in the Ley Student Center.