

1 CSCE 206—Scientific Applications Programming

This is not a course in “programming.”

This is a course in “computing.”

We will do a lot of programming.

This will seem at times like a course in programming because we will sometimes get bogged down in details, and there are **lots** of details in programming.

Try to remember

- that this is a course in “computing;”
- that to know “computing” you must know some “programming;” it is the vehicle that gets us to the destination, not the destination itself;
- that our goal is to know about the computing of **numerical** applications.

Numerical Applications

Top 500 list www.top500.org

roots of equations

interpolation of functions

fitting equations to data

numerical differentiation

numerical integration

matrix operations

averages, deviations, moments, other statistics

differential equations

Each of these is a research discipline unto itself. We will only skim the surface of the applications.

The Basic Problem in Numerical Computation

We have have a mathematical function described

- as the function that runs through a set of data points;
- as the “average” of a set of data values;
- as the area under a curve;
- as the simultaneous solution of a set of equations;
- as the function whose *changes* in value can be described;
 - distance travelled given velocity and acceleration;
 - tomorrow’s temperature given today’s temp, windspeed, humidity, etc.

The naive view that we will take is to represent all functions as polynomials. We can compute with polynomials, we can differentiate polynomials, the arithmetic is predictable, etc...

Of course we will not get results as accurate as if we had use more adult methods, but many adult methods are merely refinements of the simple ones. You will learn the basic concepts, but not the fancy details.

Why Program in C?

Why not Matlab? MathCad? Maple? Mathematica?

For “small” applications, these are all fine.

For “real” applications, they can be slow and inflexible.

“Real” computing requires a real language.

Alternatives

C++ – somewhat larger and more clumsy than C

Java – not really suitable for numerical applications

Fortran – becoming logistically difficult to teach

Remember also that the goal is to teach “computing using a programming language” and not to teach the tool that lets you program in C. If you learn a tool, your knowledge will be obsolete within three years. If you learn to program and to know how tools work, you will be able to adapt later to use different tools.

Why Unix?

“I view different computer operating systems as being like different cuisines. Using an Apple is like keeping kosher; the believers would not live any other way, but they cannot eat with members of other religions. Using Unix is like preparing your own meals from recipes in *The Joy of Cooking*; the effort involved initially exceeds the palatability of the results, but experience eventually brings satisfaction. Using Microsoft Windows is like eating at a McDonald’s; you can find one anywhere, and the food will keep you going, but it would be sad if there were no other restaurant in town.”

Gerald Folland, *American Mathematical Monthly*, March 2000.

Which Unix?

It is almost true that all Unix is identical.

- BSD Unix
- Linux
- Solaris (Sun Microsystems)
- HPIX (Hewlett-Packard)
- Irix (Silicon Graphics)
- UNICOS (Cray Research, Inc.)
- AIX (IBM)

There was once a split between “commercial” unix and “free” unix, but now it seems that all versions are converging toward Linux.

Bottom Line—Scientific Applications Programming

We will cover

- computing
- using C as the programming language
- on numerical applications
- on a standard unix platform.

You will be doing computing as it is done in the major leagues (Los Alamos, NCAR, Livermore, etc.)

You will learn enough about programming to be able to translate from C to some other language as needed.

You will learn enough about computing to be able to change to another platform and tool if needed.

The **operating system** (in this case Unix)

The **development environment**

This includes the **editor** (which you are free to choose)

We will not use a **GUI** (graphical user interface)

We will use a **command line interface**

We will use a **makefile** for convenience

The **compiler** will translate **source code** into **object code** and then the **linker/loader** will convert object into **executable**.

(Actually, you almost won't notice the object code and the linker/loader; you will mostly think the compiler goes directly from source to executable.)

We will also use **libraries** (math functions, etc.)

CSCE 206—Local Rules for Writing Programs

I don't use Blackboard. We have a CSE secure web server that has all the needed capability.

There is a CSE dropbox you can get to from the secure server.

The secure web site allows you to redirect your email from your CSE account to whatever other account you wish to use.

There is a group mailing list `csce206-001@cse.sc.edu`

I will by default send email to your `whoami@cse.sc.edu` account and only to that account.

Some Thoughts on Programming

“Lesser artists borrow; great artists steal.” (Stravinsky)

There is a great deal of code available to look at and use.

From whom may you steal?

- From yourself—from one program to the next—YES
- From the text—from all the sample code—YES
- From the internet—NO
- From yourself—from one program to the next—YES
- From other reference texts—NO
- From the text—from all the sample code—YES
- From the internet—NO

There are programs that I can use to compare one program against others for similarities.