Introduction

Why learn the C language?

Because the C language is like Latin – it is finite and has not changed for years. C is tight and spare, and in the current economic climate we will need a host of young people who know C to keep existing critical systems running.

C is built right into the core of Linux and Unix. The design idea behind Unix was to write an operating system in C so all you needed to port it to a new architecture was a C compiler. Linux is essentially the success story of a series of earlier attempts to make a PC version of Unix.

A knowledge of C is now and has been for years a pre-requisite for serious software professionals and with the recent popularity and maturity of Open Systems this is even more true. The terseness and perceived difficulty of C saw it being ousted from university teaching during the late 1990s in favour of Java but there is a growing feeling amongst some teaching communities that Java really is not such a good place to start beginners.

Students paradoxically arrive at colleges knowing less about computing than they did ten years ago as programming is seen as too difficult for schools to teach. Meanwhile the body of knowledge expected of a competent IT professional inexorably doubles every few years.

Java is commonly taught as a first language but can cause student confusion as it is in constant flux, is very abstract and powerful, and has become too big with too many different ways to do the same thing. It also is a bit "safe" and insulates students from scary experiences, like driving with air-bags and listening to headphones so you take less care. The core activity of writing procedural code within methods seems impenetrable to those who start from classes and objects.

So where do we start? A sensible place is "at the beginning" and C is as close as most of us will ever need to go unless we are becoming hardware designers. Even for these students to start at C and go further down into the machine is a good idea.

C is like having a very sharp knife which can be dangerous, but if you were learning to be a chef you would need one and probably cut yourself discovering what it can do. Similarly C expects you to know what you are doing, and if you don't it will not warn before it crashes.

A knowledge of C will give you deep knowledge of what is going on beneath the surface of higher-level languages like Java. The syntax of C pretty-well guarantees you will easily understand other languages that came afterwards like C++, Java, Javascript, and C#.

C gives you access to the heart of the machine and all its resources at a fine-grained bit-level.

C has been described as like "driving a Porsche with no brakes" – and because it is fast as well this can be exhilarating. C is is often the only option when speed and efficiency is crucial.

C has been called "dangerous" in that it allows low-level access to the machine but this scariness is exactly what you need to understand as it gives you respect for the higher-level languages you will use.

Many embedded miniaturised systems are all still written in C and the machine-to-machine world of the invisible internet for monitoring and process control often uses C.

Hopefully this list of reasons will start you thinking that it might be a good reason to have a go at this course.

References

The C Programming Language – Second Edition – Kernighan and Richie
ISBN 0-13-11-362-8

The GNU C Library Free Software Foundation C Manual
http://www.gnu.org/software/libc/manual/

MySQL C library
http://dev.mysql.com/doc/refman/5.5/en/index.html

The GD C library for graphics
http://www.libgd.org/Documentation

APXS – the APache eXtenSion tool
http://httpd.apache.org/docs/2.2/programs/apxs.html

Apache
http://httpd.apache.org/docs/2.2/developer/

"The Apache Modules Book" Nick Kew, Prentice Hall
ISBN 0-13-240967-4

A **Source Code Zip File Bundle** is supplied with this course which contains all the material described and a Makefile.

The teaching approach

I began university teaching later in life after a career programming in the telecommunications industry.

My concern has been to convey the sheer fun and creativity involved in getting computers to do what you want them to do and always try to give useful, practical, working examples of the kinds of things students commonly tell me they want to do.

Learning a language can be a dry, boring affair unless results are immediate and visible so I tend to use the internet as the input-output channel right from the start.

I prefer teaching an approach to programming which is deliberately "simple" using old-fashioned command-line tools and editors and stable, relatively unchanging components that are already built-in to Unix and Linux distributions such as Suse, Ubuntu and Red Hat.

This is in response to the growing complexity of modern Integrated Development Environments (IDEs) such as Developer Studio, Netbeans and Eclipse which give students an illusion that they know what they are doing but generate obfuscation.

My aim is to get students confident and up to speed quickly without all the nightmare associated with configuring complex tool chains. It is also essentially a license-free approach and runs on anything.

With this fundamental understanding about what is really going on you can progress on to use and actually understand whatever tools you need in your career.

In order to give a sense of doing something real and useful and up to date, the focus is on developing visible and effectively professional-quality web-server and client projects to put on-line, using:

Apache Web server and development libraries.
C language CGI programs (C programming using the "make" utility).
C language Apache modules.
MySQL server with C client library interfaces.GD graphics library with C interfaces.
Incidental use of CSS, (X)HTML, XML, JavaScript, Ajax.

This course has been designed for and lab-tested by first and second year Computer Science Students at Kingston University, London UK.