Contents

Reports & Opinions

Reports

Dialogue

Features

Reviews

Copy Dates

C Vu 16.5: September 7th

C Vu 16.6: November 7th

Contact Information:

Editorial: Paul Johnson
77 Station Road, Haydock,
St Helens,
Merseyside, WA11 0JL
cvu@accu.org

Advertising: Chris Lowe
ads@accu.org

Treasurer: Stewart Brodie
29 Campkin Road,
Cambridge, CB4 2NL
treasurer@accu.org

ACCU Chair: Ewan Milne
0117 942 7746
chair@accu.org

Secretary: Alan Bellingham
01763 248259
secretary@accu.org

Membership Secretary: David Hodge
01424 219 807
membership@accu.org

Cover Art: **Alan Lenton**
Repro: **Parchment (Oxford) Ltd**
Print: **Parchment (Oxford) Ltd**
Distribution: **Able Types (Oxford) Ltd**

Membership fees and how to join:

Basic (C Vu only): £25

Full (C Vu and Overload): £35

Corporate: £120

Students: half normal rate

ISDF fee (optional) to support Standards work: £21

There are 6 issues of each journal produced every year.

Join on the web at www.accu.org with a debit/credit card, T/Polo shirts available.

Want to use cheque and post - email membership@accu.org for an application form.

Any questions - just email membership@accu.org

Reports & Opinions

Editorial

Another end to another academic year and the usual cohort of students have passed through the doors of the country's universities with their pieces of paper signifying the sum of their past three or four years of study. For some, it's their first step into the wide world of employment, for others – who can say. Sounds nice doesn't it?

The problem really is that there is a world of difference between the world of academia and the world of business and unfortunately for the students, they don't seem to realise this. Being up and in work for 9am is not normal, neither are long vacations or the excuse of having roadworks in the centre of Manchester enough to cut the ice.

For business there is a problem not only with this transition, but also when interviewing.

At the April conference, I was talking with a very nice chap about this problem. He cited one example (the waiter in the restaurant problem) which required a threading model. Now that in itself is not a big hassle, the difficulty comes in which threading model is used – the posix implementation or the one used by Microsoft? Typically, universities use the Microsoft.NET packages, therefore use the MS implementation and are not generally taught the posix model. Without knowing the posix model and without the employer specifying which model to use, the prospective employee really only has at best a 50/50 chance of using the correct threading model.

Is this the fault of the employer or the university? I would suggest it's six of one, half a dozen of the other. Yes, it is the responsibility of the university to teach both threading models (and to use both), but there is then the problem of having to have more than one compiler per machine. Not a huge problem, but given most machines work on a standard ghost image, this does mean additional problems when installing. There is also the time factor – a typical 3 year course really only lasts 18 months which is simply not enough time to include a second threading model without the detriment of another aspect. For the employer, it's both not talking to institutions about requirements and also possibly not specifying which thread model to use; a Unix (or variant) company will have almost no use for someone using the MS model!

The Importance of One

Eh?

By the time this edition hits the doormat, the free implementation of the .NET framework, Mono, will be at version 1.0 and released to the waiting masses.

Mono gives all supported platforms access to the C# language and everything that .NET offers. How is this possible?

Well, unlike Sun, Microsoft decided to make the API for .NET open and have a published standard for the language. This has meant that anyone who wishes to sit down and implement the API will have access to this rather good language. I must admit that I didn't like C# originally (being more a cross over language than a distinct language – or so I was led to believe), but having used it both for fun and to review books with, I am very impressed with Mono and the C# language.

There are versions for MacOSX, Linux, Unix, BSD, Windows and quite a few others in development at `http://www.go-mono.com`

More Changes....

There is yet another new series starting . This time it's Objective C.

Please, don't think that just because there are lots of articles already in C Vu, that we're not after anything for a while, that's not the case. If you have something you think is up to being in C Vu, please send it in. Currently, I'm looking for a good introduction to C++ as well as material on Java and over on the Overload side of things, I know Alan is screaming for material!

A Diversion...

There has recently been quite an interesting discussion on the `accu-general` mailing list regarding the temperature at which water boils at for a given pressure. This is right up my street having been a physical chemist in a previous regeneration.

Water, as we all know, boils at 100°C (1 atmosphere pressure) – perfect for a cup of tea. If you were to go up a mountain though, the temperature at which the water boils drops, sometimes as low as 75.5°C (roughly 8000m above sea level) – absolutely useless for making a cup of Earl Grey or Darjeeling.

There is a dependency therefore between the height (and therefore pressure) and the temperature water boils at. This has caused about a week's worth of debate. Anyway, I'll provide the answer for someone to provide the solution as a program. The shortest gets a prize out of the editor's lucky bag.

How to work it out though?

The answer is quite simple.

At ground mark, atmospheric pressure (in mm mercury) is 29.921 and as you move up from ground level, the pressure will decrease. This decrease can be calculated using the formula:

```
Pressure (in. Hg) =
  29.921 * (1-6.8753*0.000001
    * altitude, ft.)^5.2559
```

(these are from really old notes given during my degree, which is why I'm using inches of mercury instead of standard atmospheric units and feet instead of metres).

We know (from what we've seen) that as you move up into the atmosphere, there is a decrease in the boiling point of water. Again, it's quite simple to work this out:

```
Boiling point = 49.161
        * ln (in. Hg) + 44.932
```

(this is all perfect spreadsheet fodder)

Again, this is in °F rather than °C, so:

```
C = 5/9 * (temp in °F – 32)
1m = 3.282 ft
```

What the actual numbers represent have been lost in the mists of time (they had a nasty accident with some Old Peculier a few days after my finals!)

What you should be able to do now is transform this into a quick and simple program (you choose the language). Happy hacking!

A Cautionary Tale

I've had quite an interesting week this week.

Besides the usual rush to have the magazine ready for putting to bed (for which I'm sure we should all be charging our respective glasses to the sterling and hard work of our Production Editor who doesn't seem to get anywhere near as much credit as she deserves), I had an unusual phone call from somewhere that I had applied for a job at a while ago.

While that may not seem odd to quite a few people, it was odd to me as normally in the education sector, if you don't get the job, that's it – you don't hear anything back. This was unusual as it was to give me feedback on exactly why I didn't get the post.

Now, despite knowing everything required and being possibly the best candidate for the position, the reason for not getting the post was that the employer had done a web search on `groups.google` to see what could be seen.

On usenet, I have quite a high profile (and not just on the programmers groups) and unfortunately have been involved in a very small number of flame wars – one of which happened last year on the `uk.comp.os.linux` group. Now, as this is actually a matter of some ongoing legal action against the chap who started it all (and also a long running Police investigation into this character's nefarious activities), I can't comment on the nature of the flame, but because of it it the company decided to dig further... and further... and further.

In one sense, it actually helped my application as it showed the nature of how I work with a disperse group and the methods I employ to solve problems. On the other hand, the number of open source groups I contribute to made the company consider me possibly not the best candidate for proprietary work.

Let this be a lesson then: while usenet is fun and a very useful forum for learning on, it can act as a double edged sword.

Now, Applying That Here...

But how does this apply to the us, the ACCU? Well, we do have a number of publicly open groups where not only members of the ACCU reside, but also prospective employers and those who just need help. It is therefore of paramount importance that a professional attitude is given and threads on the list kept on topic. All too often a thread begins with something sane, but by about 10 replies down the line, the original topic has changed, but the email subject title hasn't – very confusing and if you get the same number of emails as I do, then the probability of missing something useful increases.

It also demonstrates an important difference that should be recognised between a personal and professional persona.

For instance, it would be unprofessional for me to post from my ACCU account and be expressing a detrimental opinion against any particular vendor of any particular product as it

could be considered as being that of the ACCU; something which may definitely not be the case. Now, the case is different if I post from my personal email address. Many who subscribe to the `accu-general` list will know my opinions of London, it definitely isn't endorsed by the ACCU or anything like that, but for some, the difference between me as the editor and me as, well, me is not always clear. Which is another problem with people when scanning usenet!

Enough of that ramble, many will be puzzled as to why I brought it up. The answer is simple.

One fund raising idea the committee has been considering is "selling" ACCU email addresses (in the same way as is done with the ACM), so it would be possible to have the email address `paulf.johnson@accu.org` – sounds nice, but having such an email address would put a distinct number of problems on us which would need a disclaimer to be added. However, how many bother reading disclaimers?

We have a standard one at work

> "Any views or opinions are solely those of the author and do not necessarily represent those of the University of Salford unless specifically stated. This email and any files transmitted are confidential and intended solely for the use of the individual or entity to which they are addressed."

Would that be enough if we were to make available ACCU email addresses? I personally don't think it would be.

This idea is still being discussed, but I'm sure your opinion would be greatly appreciated. If you have comments, please send them to the committee.

An ACCU email address and its inherent problems are not the only aspect where this distinction between ACCU-endorsed and not endorsed material raises its head. The other place is obviously in the book reviews.

A claim levelled by some book companies is that as we publish the reviews, they therefore must be the views of the ACCU. Again, this is not the case; all reviews are a personal opinion of the reviewer. This is definitely a problem with respect to the website. While we have a limited readership of the printed magazine, the book reviews are publicly available for all to see without any form of disclaimer as to the personal nature of the reviews. You can now appreciate the problem of a disclaimer...

Paul F. Johnson

View From the Chair

Ewan Milne `<chair@accu.org>`

Just a brief report from me this issue. This does not reflect a lack of activity by the committee: far from it, we are currently involved in a review of our system for book reviews, arranging for the long-awaited website revamp, and getting underway with the development of next year's conference. But all these are ongoing tasks still to reach fruition, and so more details on each soon.

You may have noticed that the last issues of C Vu and Overload (16.3 and 61 respectively) coincidentally marked the arrival of two new editors.

Paul Johnson was elected C Vu editor at this year's AGM, and John Merrells was elected Publications Officer: John's first action in his new post was to appoint Alan Griffiths as his successor in the Overload editor's chair. I'd like to welcome both new editors to their jobs, undoubtedly both key roles for the association. The life of editor of an ACCU journal can be stressful, but luckily there is something we can all do to alleviate this – write and submit articles! Both Paul and Alan will be grateful for your submissions. If you have never previously written and are concerned about producing a piece of high enough quality, remember that there is an excellent editorial support system in place to carry articles from draft to publication. All our regular contributors will tell you that it is a most fulfilling exercise.

Standards Report

Lois Goldthwaite `<standards@accu.org>`

The future of C++ is already taking shape. The international C++ standard committee, WG21, is stepping up the workload as they move toward a revised standard, still several years in the future. A visible sign of this is that the committee are adding two more "mailings" of documents per year to the four existing ones. (They are still called mailings even though the days of shipping paper copies around the world are long gone.) The additional mailings are scheduled for July and January, midway between the committee's face-to-face meetings in April and October every year. This gives committee members longer to study the issues and solicit comments from the public.

Nearly all committee papers are publicly available on the WG21 website at `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/`. An exception to this policy is documents relating to the C++/CLI effort in an ECMA technical group – those are password-protected. However, the regular policy does apply to the current working draft of the C++ standard itself. This draft incorporates not only Technical Corrigendum1 (the version available at bookstores in the Wiley edition) but also any defect reports resolved after TC1 was voted out. The committee has no plans to issue a second TC.

Speaking of C++/CLI, it appears that the schedule for this document is slipping. This is partly because it is closely tied to a revision of the ECMA Common Language Infrastructure standard (more commonly known as .Net), and the people working on that document have decided it will not be ready in time to meet ECMA's fall deadline for a vote this year. Correctness and consensus take time to mature, even in a very small group of experts working under a very lightweight process.

The international C committee, WG14, have reconfirmed the C standard without intention to revise it, although they do plan to issue a second Technical Corrigendum at some point. They have decided that TC2 will be issued in the form of an integrated document, as the C++ TC1 was. (The Wiley edition of the C standard does feature its TC1 integrated into the 1999 standard, but ISO originally published the revisions as a small separate document. BSI have long had a policy of integrating such changes and issuing a complete version.)

Membership Report

David Hodge `<membership@accu.org>`

The final membership total for this year was 1099, 26 down on the same time last year. It is renewal time, so please be aware that there is a subscription increase, see the June C Vu for details. Please make sure that you renew in time as under the new system we no longer distribute back issues. You should be renewing by the 31st August.

If you renew after 30th September you will not receive the October issue. You can however get access to the journals on the website.

Letter to the Editor

Not exactly a huge mail bag this edition, but this interesting one came in from Francis Glassborow.

A New Publishing Opportunity

I recently spent time with Parchments, ACCU's printers, discussing how their technology is changing and what those changes enabled.

One of the dramatic changes is that they can no produce single copies of books in much the same way that you might previously have asked them to produce a single copy of a poster. The feature that I found interesting was their estimated cost for a 100 page 'perfect bound' book with a full colour laminated cover. This was substantially under £4 for any production run between 1 (yes a single copy) and 150. After that the price goes down because a cheaper option cuts in for reproducing the pages.

This has set me thinking. Probably many of you have a book that you could write, probably not a best seller and probably not one that a major publisher would contemplate, but nonetheless a good book.

There are a number of processes in going from an author's grey matter to a delivered book and quite a few authors make a mistake by under-rating the added value of a good editor, good design and layout, careful copy-editing etc. Then there are issues like ISBNs and the requirement to provide (in the UK) copyright copies to the British Library.

Now I am well familiar with most of those issues. On the other hand I really do hate administration. I wonder if any of you would be interested in exploring creating a partnership or small company to specialise in producing very high quality (validated technical content, properly copy-edited and professional standard of layout and production) short run books. I have thought about many of the issues but won't take up ACCU's space by going into them here. If the idea interests you please contact me (+44 (0)1865 246490 or `francis@robinton.demon.co.uk`)

And if you have an idea for a short (80 to 200 page) book you would like to write and have published let me know about that as well.

Francis Glassborow

It certainly looks an interesting idea.

As always, should you have any comments or other items of interest, please don't keep them to yourself – send them in. I'm always happy to receive your views.

Paul F. Johnson

Dialogue

Francis' Scribbles

by Francis Glassborow aka 'The Video Guy'

Time for Change

When I picked up the current issue of C Vu, I was surprised to say the least by the item on the inside back cover. Over its seventeen years of existence ACCU has very rarely changed its membership rates. When ACCU was first founded as CUG(UK) the cost was £10 for six issues of the newsletter. A few years latter when we had stabilised to an annual membership fee and a guaranteed number of issues of C Vu per year the cost went up to £12. Several years latter when we re-organised to two levels of membership and added a Corporate membership the costs went to £15, £25 and £80. Since then careful housekeeping and the acquisition of substantial advertising revenue has kept the costs at those levels despite going to full colour covers, professional production and so on.

Advertising revenue is fine but it does require someone with considerable expertise to bring in and hang on to advertisers. This is true for all publications. Getting advertising is hard work.

A second quiet change has been happening. The growth of ACCU over the last ten years has been almost entirely in non-UK membership. The cost of postage has steadily increased. Originally the contribution made to general administration costs by non-UK membership fees was low but positive. By that I mean that if we worked out the cost of the printing and distribution of C Vu and Overload to non-UK members it was only a little less than they were paying in membership. None of us had any concern about that because we believed in the principle that a truly international organisation should not have differential fees depending on geographical location.

With the steady increase in production and distribution costs for our periodicals, overseas membership has moved from marginally in the black to substantially in the red. Couple that with a very welcome increase in non-UK membership (over 40 countries the last time I looked) and the loss of advertisers and we can all see that a substantial readjustment of membership fees became necessary.

Now there are two things you can do to help keep ACCU membership fees stable for another decade. First you can help increase the number of members. That allows the administration overheads to be spread over more people. The second thing is to think carefully about how to bring in more advertisers. In days gone by my rule of thumb was that selling all six cover pages should pay for the professional production editing of our periodicals. Other advertising should be such that a page of advertising pays the costs of two pages of editorial content. That would mean that an issue of C Vu with 32 full pages of editorial content would be fully paid for if all the three cover pages were sold and an extra 16 pages of advertising were included. To actually achieve that you would need to pay a full time advertising manager so it will not happen. Nonetheless every little bit helps.

So what has this to do with the title of this item? Well it was seeing that change in membership fees that started a train of thought. Things are not immutable. We need to take stock from time to time and make necessary and purposeful changes. What we should not do is change for change's sake. We should always strive to understand why things are the way they are, and understand what we are trying to achieve.

I am reminded of the very first essay in *Programming Pearls* (Jon Bentley, 0-201-65788-0) which should be required reading for everyone who is asked questions of the form 'How do I do...' Overtly the essay is about sorting but that would be a very shallow view. The point that Jon Bentley was making is that we should be wary of answering such questions with anything other than 'Why do you want to do that?' Until we have the answer to that question, even the most erudite direct answer is unlikely to actually help.

Now go back to what I have written above and see how, I hope, I have applied that lesson to the question of membership fees. There is a lot more that I have not written, but the essence is that reactions to changes to the membership fees must be based on an understanding of what they are for and why ACCU needs more income.

If all that our Committee did was to up the fees they would be doing a poor job but you know as well as I do that they are one of the most hardworking committees of any purely voluntary organisation. Quite apart from keeping ACCU running on a day-to-day basis they are also reviewing many aspects of ACCU. In many cases these are things that have just grown out of an accumulation of small decisions that made sense at the time.

The nature of ACCU has changed slowly but surely. In the early days the Committee was almost entirely composed of enthusiasts and there was a fair sprinkling of amateurs (those for whom programming was no part of their paid work). These days the ACCU Committee is composed almost entirely of professional developers with a sprinkling of language experts.

Each year a number of people resign from membership because they have moved on from programming. Some of those have genuinely moved out of IT and no longer have any interest in software development. However for a good number this is not the case, they have simply moved up the hierarchy to jobs that do not involve expertise in use of one or more computer languages.

Now the point I want to put to you is whether ACCU should expand so that those longer-term members whose careers have developed still have a place in ACCU.

I feel the answer should be yes but I am far from certain that I know how we could achieve that. Of course there is a way in which such issues are entirely academic for me but that does not prevent me from raising the question and pondering about an answer.

Book Review Classification

One of the things that has grown by accretion is our book review system. I think it is past time that we gave it a good shaking and decided the degree to which it should change. To do this we must focus on why we review books and why we add a classification to reviews on the website.

A book review carried out by a single reviewer is always a personal statement by that reviewer. As a reviewer I can choose to recommend a book or tell you that I think you should leave it firmly on the retailer's shelf. It is very important that publishers of reviews are open to publishing second reviews that are in radical disagreement with the first. The publisher must also willingly withdraw any statement that is factually incorrect.

Now as the number of reviews grew and we started publishing them on the web we started adding a recommendation. This was intended to help people who lacked time to read all the reviews by steering them towards ones they might find worth consideration. Unfortunately, as time has gone by these recommendations have become increasingly perceived as either an ACCU one or that of the reviewer (which sometimes they were). I find that very dangerous. And it is time for change.

However my view is that in that change we need to make it much clearer that the any recommendation is that of a specific reviewer and not and ACCU one. The reviews on our website are not and never have been ACCU reviews. If we wanted to do that we would need a review panel and a consensus developed before we published a recommendation. For books that are already published we would never have the resources even if we had the will.

For now I want you to think about the issues around this one very public area of ACCU's work. Should ACCU as a body ever endorse a book? If so, how should we do it? In coming to an answer we must first understand what we are doing. We must also understand what we can reasonably deliver.

I have some very specific ideas about this which I will present in a later column.

Another Portable IDE

While searching for a multi-platform IDE for my next book I came across JGrasp. This is an IDE explicitly developed for teaching purposes. The developers are a group at Auburn University. While the implementation is in Java and the original work was done for assist with teaching Java, it is multi-lingual as well as multi-platform.

It supports a considerable range of C++ compilers and has some very nice features. At the moment there are a number of flaws on the C++ side. For example, it does not currently have a simple way to give both a simple library path and enable use of third party libraries. It is fairly simple to modify the scripts to solve that problem, and in the next release that will have been done.

In the meantime, if you have a moment have a look at it from http://www.jgrasp.org/ and let me know your thoughts about it.

And while I am thinking about it, can someone explain why G++ has that horrible way of modifying file names on the command line (adding 'lib' to the start of a library name to determine the file name.)

Other Periodicals

Over the last few years there has been a terrible decimation of periodicals for software developers over the last few years. The highly specialised ones such as those for the embedded software developer have managed to hang on in their niches. Dr Dobbs' Journal has survived the general carnage but the breadth of its coverage means that for most readers only a few items are of potential interest in any single issue.

CUJ (The C/C++ Users Journal) traces its origins back to being the newsletter of the C User Group (a US based group that was founded a little before ACCU – originally called CUG(UK) though it had nothing to do with the US group). While CUG(UK) developed into the ACCU we have today, CUG became no more than the tail on the CUJ body. Our periodicals serve ACCU members whilst what is left of CUG is a small added value for CUJ subscribers. I have not seen a copy of CUJ for some time but understand that various changes have happened over the last couple of years which have culminated in Chuck Allison departing as editor (and I note that Bill Plauger is no longer listed in its editorial staff). If anyone can provide details, or better still write a review of CUJ as it is in 2004 I would be grateful.

Now those who went to Chuck's Keynote at this year's conference will know that he is heading up a new electronic publication specifically for C++. That has now gone live and you can see what is happening by visiting http://www.artima.com/cppsource/

Now from the newest to the oldest (at least I think it is). *Software Practice and Experience* is currently in its 34th year of publication. Like DDJ, it covers a very broad range, unlike DDJ it is a genuinely peer reviewed 'academic' publication. Unfortunately it is extremely expensive. Through the early 90s it tended, in my opinion, to be too academic in the prose style of its contributions. This resulted in thousands of words of turgid prose whose aim seemed to be to hide great information behind text that did everything but assist in communication. Either this has greatly improved over the last few years or I have become better able to handle the academic prose style.

A New Sort Algorithm

The last of the four papers in the current issue of SP&E (Vol. 34, No 8) concerns a new sort function for the C Library. This is a generally excellent paper on a sort algorithm with a performance of O(n log n), I was irritated by the author's lack of understanding of what the C Standard actually requires of its `qsort()` function. The author seems to make the common, but erroneous, assumption that `qsort()` implements some variation of Hoare's Quicksort. It does not. I have little doubt that many implementations of the C Library do in fact use Quicksort but nowhere does the Standard require that to be the case. Actually the author seems to assert that `qsort()` will normally be the Bentley and McIlroy modification of Quicksort.

As I read through the paper my irritation grew. As the opening sentence of the section titled 'Conclusions' begins *'So far, all sort library functions have been based on Quicksort, …'*. Such assertions have no place in an academic paper.

Why does this matter? Well apart from perpetrating an error it also might lead people to believe that a Standard C Library could not use the author's (Jing-Chao Chen) Proportion Extended Sort. However a library implementor can use any sort that meets the very limited criteria provided by the C Standard. C++ is rather more demanding in its requirements for `std::sort()`, however even those have been outdated by the development of a hybrid sort in the late 1990s.

I have no doubt that Proportion Extended Sort is a worthy addition to the catalogue of available sorting algorithms and that generally an implementor would be advised to select it in preference to any of the variants of Quicksort that are frequently used to implement `qsort()`. Knowing that it exists enables ordinary programmers to point to it when asking for a better library implementation. If you are interested you can get the code from: http://www.dhu.edu.cn/dhuwangye/kxyj/psort.htm

However I should warn you that the code is not exactly the kind of portable code that I would expect from a fully competent C or C++ programmer.

Commentary on Problem 15

Here is the code again:

```
#include<iostream>
#include<cstdio>
using namespace std;
main() {
  int n;
  int waste;
  char name[51];
  cout << "Enter any integer number...\n";
  cin  >> n;
  cout << "Enter your name...\n";
  cin  >> waste; // 'gets' does not read the
            // name if this line is omitted.
   gets(name);
}
```

Experienced C++ programmers will immediately spot the problem; the programmer has hacked out a solution to it. The code mixes different forms of access to the standard input stream (aka, console input). After getting the value for `n` there will be, at a minimum (unless the programmer uses that horrible 'Ctrl Z' for Windows or 'Ctrl D' for Unix (and variants) which is, in my opinion, one of the few blots on *Accelerated C++*) a newline character left in the input buffer. Using any of `gets()`, `fgets()` or `getline()` will read that character and stop.

The hack that the programmer has come up with is to try to read a number. Now that read will succeed if the user carelessly types in a number before entering their name on the same line. Or it might fail because the next non-whitespace character read from `stdin` is not a digit or a plus/minus sign. However whichever happens (as long as the number isn't on the same line as the integer entered for `n`) the newline character terminating the previous input has been consumed.

Now either `gets()` or `fgets()` will correctly read the following whitespace terminated entry. However all versions of the C++ `getline` will fail unless the user actually did provide `waste` with a numerical value. The reason being that `std::cin` will now be in a fail state and so ignore all input requests.

The positive aspect of this example is that the original programmer had the sense to ask why his hack worked. However the warning is that learning to program is much more than just getting code to compile and produce the result you expect. It is essential that the programmer understands why the code works.

Problem 16

Comment on the following both as Java and as C++.

Have a look at the following tiny function. The problem is insidious; the same code is legal in Java and does exactly what you want, while in C++ it compiles without error.

```
string to_string(int n) {
  if(n == 0) {
    return "NULL";
  }
  else {
    return "" + n;
  }
}
```

Cryptic clues for numbers

Last time I gave you:

Sounds like a perfect result when a score dine together. (2 digits)

Perhaps it was too tough for most of you, as I have had no responses. Perhaps the clue needs a bit more polishing. The answer is 28 which is the second perfect number (a number which is the sum of all its proper divisors; 28 = 14 + 7 + 4 + 2 + 1). Aloud that sounds like 'twenty ate'. However the positioning of the 'sounds like' in the clue is wrong. Perhaps a better version would have been:

Sounds like a score dining together was a perfect result.

Taking a basic idea for a clue and honing it takes both time and experience. The latter I have but the former was lacking last time. My apologies.

Now, try this one:

Looking to two fat ladies for a solution? Too gross!

When you have the answer see if you can provide either a new clue or improve my one. As an incentive I will send the author of the best clue (in my judgement) a copy of *The Elements of C++ Style.*

Francis Glassborow

Student Code Critique Competition 29

Set and collated by David A. Caabeiro <scc@accu.org>
Prizes provided by Blackwells Bookshops & Addison-Wesley

Please note that participation in this competition is open to all members. The title reflects the fact that the code used is normally provided by a student as part of their course work.

This item is part of the Dialogue section of C Vu, which is intended to designate it as an item where reader interaction is particularly important. Readers' comments and criticisms of published entries are always welcome.

Before We Start

Thanks to the helping hand given by our editor and by a member of the committee, I've been able to get my hands on the only two entries for the current issue. It's quite sad to receive collaboration from people who contribute to ACCU in many other ways, and not from you. Being a 1000+ members association, if roughly 0.5% of the members participated, there could be plenty of material to provide food for thought. Please let us change this statistic for the better.

Remember that you can get the current problem set in the ACCU website (`http://www.accu.org/journals/`). This is aimed at people living overseas who get the magazine much later than members in the UK and Europe.

Late submission to SCC 27

From Tony Houghton <h@realh.co.uk>

Let's start by solving the immediate problem. If `pf()` encounters a prime number it returns an "empty" array i.e. the first element is zero. This means that the body of the `while` loop in `main()` is not executed and the `ugly` flag is not altered; it retains its value from the previous number in the range, which is often ugly, at least for low ranges. The solution is therefore to reset the `ugly` flag to zero in each iteration of the enclosing `for` loop. Immediately before or after the `line idx = 0;` is ideal.

However, the code is still not performing the correct test. It only proves whether or not the last factor in the array is ugly, not that they all are. This may work with the defined set of ugly factors and because of the order in which `pf()` fills its arrays, but we should rewrite the test instead of relying on this. It's easiest to prove that a number is not ugly, so we'll start by assuming it is until we find a non-ugly factor, not forgetting the initial problem with primes.

But we have something else to take into account: 2, 3 and 5 are prime, so `pf()` will return an "empty" array and the test will not realise they're also ugly. We could deal with this in `pf()` by copying `quotient`'s initial value from the variable number rather than 0, but this leads to an inconsistency when the `pf()` function is considered in its own right: prime numbers have themselves listed as a factor, other numbers don't.

As we're also excluding the number 1 as a factor it makes sense to continue to exclude the number itself and explicitly check for 2, 3 and 5 in our test. Thus the main `for` loop becomes:

```
for(n = start; n < stop + 1; ++n) {
  idx = 0;
  flist = pf(n);
  if(flist[0] == 0 && n > FIVE) ugly = 0;
  else ugly = 1;
  while(flist[idx]) {
    if(flist[idx] != TWO &&
       flist[idx] != THREE &&
       flist[idx] != FIVE) {
      ugly = 0;
    }
  ++idx;
  }
  if(ugly == 1)
    printf("%d\n", n);
  free(flist);
}
```

Now to give the code a complete makeover, starting from the top and working down:

The first issue we encounter is a number of preprocessor macros. Macros should only be used for jobs that nothing else can do, but these can be replaced with `const int` and/or `enum`. Furthermore, the purpose of replacing "magic numbers" with named constants is so that if the values need to be changed they only need to be changed in one place. If we want to change this program to deal with a different set of ugly factors, the current names of the constants will be very confusing; if we don't ever want to change it the names are redundant. I've rewritten these constants as:

```
const int MaxFactors = 20;
  /* Max number of prime factors to find */
enum UglyFactors {
  UglyFac1 = 2,
  UglyFac2 = 3,
  UglyFac3 = 5
};
```

(Sorry about the pun).

Next we have a prototype for `pf()`. The name is far too terse, let's make it more descriptive: `prime_factors()`. The function is only used in the same source file, so it should be declared `static`. I'm not sure whether the student has covered linkage types yet, but I think it's a relatively straightforward concept and a good habit to learn early.

It appears to be common practice to declare all function prototypes in advance and define static functions towards the end of a file. However, I prefer to avoid separate prototypes except in headers on the grounds that they are a form of repetition. Therefore I've brought the body of the function forward to this point. The comment that precedes it should highlight any points of interest about its parameters and return value – in this case that it returns an array that's been allocated on the heap and that the array is zero-terminated.

Inside the function I've separated the variable declarations because `int*` and `int` are actually two quite different types and it's considered bad form to make them share the declaration with commas. I also prefer to give all variables their own distinct declarations unless two or more are closely related and have no initialisers.

I also decided to use the variable name `i` instead of `idx`. There is an argument for short variable names as well as long descriptive ones, and while `idx` was a good name to start with, being concise but still meaningful to almost any programmer, I always use `i` for the (primary) array index in loops myself and sticking to that convention makes the code more readable to me.

I've changed the `while` loop condition to `(divisor <= number)`; I think it's less clutter than the original. But we should also check that we haven't exceeded the array bounds. If we really want to know all the prime factors we should extend the array when necessary or come up with a rough figure that's guaranteed to be an overestimate – e.g. the upper limit of the range given to the program – and make the array this size in the first place. However, for this purpose it's adequate to return when the array is filled, remembering to leave room for the terminating zero – which does not need to be inserted explicitly because of the use of `calloc` instead of `malloc`. The complete condition is now `(divisor <= number && i < MaxFactors - 1)`.

I noticed that it's possible for the same factor to be entered more than once in the list if its square is also a factor e.g. 2 will appear twice if `number` is 8 or 12. We can prevent this by checking whether the previous entry is equal to the one we're about to add – remembering that if `i` is 0 we mustn't try to check `element-1`:

```
if(i == 0 || flist[i - 1] != divisor)
  flist[i++] = divisor;
```

I also have a rule about braces around single statements such as a simple `if` or `for` body. Although these are optional I often include them, especially if:

1. the parent statement, e.g. an `if` conditional, is long and I've wrapped it to fit my editor window – the indentation makes it difficult to distinguish the conditional from the body otherwise
2. it's an `if` or `else` clause and its sibling clause needs braces
3. there's any possibility that the statement is a macro which could be switched off e.g. an assertion or extra logging in debug builds. I mention all this because I've applied rule (2) to the `else` statement here

Next I've done some refactoring. Breaking programs down into smaller functions almost invariably makes them more readable, and providing a function to test whether a given number is ugly is a logical step. The body of this function is pretty much as above except that I replaced the `while` loop with a `for` loop. Also, bearing in mind being able to change the values of the `UglyFac` constants, it's unsafe to assume that there are no non-ugly primes below `UglyFac3`, so we should test against each of them instead of testing whether `n > UglyFac3`.

I've given `main()` a comment about its arguments, which is basically an alternative to the "enter a range" comment which I thought was misleading: it implied the range would be read from `stdin` rather than

arguments. Again I've separated the variable declarations. I've kept the variable name `n` because it's ideal for a loop variable describing a number other than an index, but I've renamed `start` and `stop` to `first` and `last`, which I think makes it clearer that the range is inclusive. The `for` loop terminating condition is again changed to use `<=` instead of `<`

Just a couple of minor semantics remaining. As `ugly` is a boolean flag, I prefer to write `if(ugly)` and `if(!ugly)` rather than `if(ugly==1)` and `if(ugly==0)`. And as we're using `EXIT_FAILURE` we might as well use `EXIT_SUCCESS` too. Here is my first complete rewrite (keep on reading for my description of a slightly different approach to the problem):

```
/* Find "ugly numbers": their prime factors
   are all 2, 3 or 5 */
#include <stdio.h>
#include <stdlib.h>

const int MaxFactors = 20;
     /* Max number of prime factors to find */

enum UglyFactors {
  UglyFac1 = 2,
  UglyFac2 = 3,
  UglyFac3 = 5
};

/* Returns a zero-terminated array of number's
   prime factors, allocated on the heap */
static int *prime_factors(int number) {
  int *flist;
  int quotient = 0;
  int divisor = 2;
  int i = 0;

  flist = calloc(MaxFactors, sizeof(int));
  while(divisor <= number
        && i < MaxFactors - 1) {
    if(divisor == number) {
      flist[i] = quotient;
      break;
    }
    if(number % divisor == 0) {
      if(i == 0 || flist[i - 1] != divisor)
        flist[i++] = divisor;
      quotient = number / divisor;
      number = quotient;
    }
    else {
      ++divisor;
    }
  }
  return flist;
}

/* Returns 1 if number is ugly, otherwise 0 */
static int is_ugly(int number) {
  int i = 0;
  int ugly;
  int *flist = prime_factors(number);
  if(flist[0] == 0 && number != UglyFac1
     && number != UglyFac2
     && number != UglyFac3) {
    ugly = 0;
  }
  else  {
    ugly = 1;
  }
  for(i = 0; flist[i]; ++i) {
    if(flist[i] != UglyFac1 &&
       flist[i] != UglyFac2 &&
       flist[i] != UglyFac3) {
      ugly = 0;
    }
  }
  free(flist);
  return ugly;
}

/* Range is given in arguments */
int main(int argc, char **argv) {
  int n;
  int first, last;
  if(argc !=3)
    exit(EXIT_FAILURE);
  first = atoi(argv[1]);
  last = atoi(argv[2]);
  for(n = first; n <= last; ++n) {
    if(is_ugly(n))
      printf("%d\n", n);
  }
  return EXIT_SUCCESS;
}
```

This solution works, but is not the most efficient. We don't actually need to list all the prime factors of ugly candidates, just check them until we find a factor that proves the number isn't ugly, again taking care not to pass primes as false positives. Therefore the `prime_factors` function can be deleted and `is_ugly()` replaced with the version below.

```
/* Returns 1 if a number is ugly, 0 if it
   isn't */
static int is_ugly(int number) {
  int divisor = 2;
  int ugly = 0;
  for(divisor = 2; divisor <= number;
      ++divisor) {
    if(number % divisor == 0) {
      if(divisor % UglyFac1 != 0 &&
         divisor % UglyFac2 != 0 &&
         divisor % UglyFac3 != 0) {
        ugly = 0;
        break;
      }
      else {
        ugly = 1;
      }
    }
  }
  return ugly;
}
```

Student Code Critique 28

Program 1

I'm newbie to C++ and I would like to know which would be the best (elegant and correct) solution for the following small (string) read problem with `gets`.

```
#include <iostream>
using namespace std;
#include <cstdio>

main() {
  int n;
  int waste; // needs this to work (read)
             // properly!!
  char name[51];
  cout << "Enter any integer number...\n";
  cin  >> n;
  cout << "Enter your name...\n";
  cin  >> waste; // 'gets' does not read the
                 // name without this line!!
  gets(name);
}
```

Mixing C and C++ functions doesn't seem the best way to write this program. Please provide alternatives, taking also into account its security implications.

Program 2

*If I enter 23 and 5 the answer should be 23 * 5 = 115 my answer is off by five or whatever number 2 is. I was told I am not including the first two numbers in the loop. I thought when I cin the numbers it is including them. Does anyone have any suggestions on how I can fix this?*

```
#include<iostream>
#include<iomanip>
#include<string>
using namespace std;

int main()
{
   while (1){
int   num1, num2, total = 0;
cout<<"Enter two numbers: ";
cin>>num1>>num2;
while (num1 != 1)
cout<<setw(5)<<num1<<setw(5)<<num2<<endl;
   num1 /= 2;
   num2 *= 2;
   if (num1 % 2 != 0)
     total += num2;
}

cout<<"the total is  "<<total <<endl;
}  //End While 1

return 0;
}
```

There are numerous errors in both the code and the student's understanding. Please address these comprehensively.

From Paul F. Johnson <editor@accu.org>

Program 1

If I ignore the obvious mistake of not having `int` before `main()`, there are a couple of problems with the code.

Firstly is the use of `gets` – it's a licence for things going wrong with undefined behaviour the most obvious to occur as a `char` array is being used to store the person's name. A far better solution would be for the name to be stored in a `std::string` variable which does not have a boundary (well, not in the same way as a `char` array does!). Buffer overflows account for more and bloodier problems with security than enough.

There is also a problem with the entry of the number – as it stands, it is possible for the user to enter just about anything they want as there are no forms of bounds or type checking.

Finally, we have the "dummy" use of another variable which isn't required (and again, can lead to problems without checking for the correct type). The stream can be cleared by use of `cin.ignore`.

By the simple application of `cin.fail()` and `cin.getline()`, the code can be transformed into something which (a) works and (b) is relatively secure.

A solution may be along the lines of

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  int n;
  string name;
  cout << "Please enter a number : ";
  while(cin >> n, cin.fail()) {
    cout << "I need a number!" << endl;
    cin.clear();
    cin.ignore(numeric_limits<
               std::streamsize>::max(), '\n');
    // non compliant compilers may complain
    // here. If they do use cin.ignore();
  }
  cin.ignore(numeric_limit<
               std::streamsize>::max(), '\n');
  cout << "Please enter your name : ";
  cin.getline(name, '\n');
  cout << "Number : " << n << ", name : "
       << name << endl;
}
```

As newbie code goes, the original wasn't that bad. It was well laid out (which made debugging simpler) and contained many of the problems faced by a newbie where the most obvious answer is not always the correct one.

Program 2

First, the easy ones...

`#include <string>` is not required and somewhere, an open { is missing. This second problem would have been obvious if the student had bothered to try to compile the code!

I next come to the `setw()` calls – what are they there for? We are dealing with integers in `num1`, `num2` and `total`, therefore using `setw` does seem to be a bit of a waste!

There doesn't seem to be a check on the entries either. There is nothing to stop the user entering # and @ for the numbers (or even CVu and Overload for that matter!).

The next couple of faults are in the logic. To start, we have the line

```
num1 /= 2;
```

If the user enters 0, the result value in `num1` will be 1 which will throw the answer.

The line

```
if(num1 % 2 != 0)
```

really is bemusing! What is the point of it? Why does the program have to check if `num1` is divisible by 2 with a 0 remainder? [*The student is trying to implement a Russian peasant multiplication algorithm. The point is to discriminate whether* `num1` *is even or not. David*]

Finally, `total` only seems to be having `num2` added to it (and even then, only as a result of a condition); `num1` doesn't seem to be used anywhere other than as an entry. It is therefore completely possible for the user to be enter as many numbers as they like without the correct answer coming out ever.

Fixing the problem is simple – it requires being rewritten. Unfortunately, the student hasn't actually said what the question was, so it is a tad pointless trying to second guess what the original problem set was.

From Mark Easterbrook <mark@easterbrook.org.uk>

Program 1

As suggested in the problem, mixing C and C++ I/O is not the best way to write this program. As you are learning C++, it is best to replace the C-type parts with C++. This requires two changes:

Change "`char name[51]`" to "`string name`" (this will require an `#include <string>` to compile). It is nearly always better to use one of the C++ collections rather than an array. In this case a C++ `string` type will automagically expand to the size of the input name so you have one less thing to worry about.

Use C++ `istream` instead of `gets()`: `cin << name`. The `waste` variable and input is no longer required, nor is the `#include <cstdio>`.

It is worth pointing out a number of improvements that can be made to the code. `main` always returns an `int` so this needs to be specified: `int main()`.

`using namespace std` will import the whole `std` namespace. It is often better to only import names you actually need, or to qualify every use.

Single character variable names are often frowned upon. Think how difficult it would be to search for all uses of the variable `n` in a large program!

At this point we have "fixed" the program and could let it rest, but it is worth looking at why `gets()` is bad, even in C code. `gets()` will read an unknown number of characters into a C string (`char*` or `char[]`) of predetermined length, therefore it is possible to read more characters than are allowed for. This is called "buffer overrun" and accounts for many of the security holes in software. The result of such a buffer overrun could be:

- It writes over memory allocated but not used by the program. No amount of testing will show this up so the bug can remain hidden until something else changes: a different compiler, a different platform, or a simple code change somewhere else in the program.
- It writes over memory not allocated to the program. If you are lucky your operating system will detect this and stop the program.

- It overwrites memory used by the program. This causes the program to malfunction, and possibly later crash. This will be very difficult to track down.
- It overwrites memory used by the program in such a way that the program does something completely different to that originally intended. If you are unlucky you are connected to the internet and this allows a stranger access to your computer!

As you cannot use `gets()`, what should you use instead? There are a number of options:

Use `fgets()` to read up to a line of input. It takes as parameters a pointer to the input buffer (like `gets`), a maximum number of characters (which prevents buffer overrun), and a file descriptor (e.g. `stdin`). It will stop reading either when the maximum number of characters are read, or at a new line character.

Use `scanf()` with the `%s` format string specifying a maximum length. For example, `%50s` will read a maximum of 50 characters (plus a terminator!).

Input characters one at a time using `getchar()` or `getc()`.

You will need to read the documentation for each of these to see which meets your needs best. Each has different rules as to when it stops reading.

Program 2

Note: The program as presented will not compile because the inner loop has a closing brace but no opening brace. This can be corrected by adding the opening brace after the while expression. I have assumed this is just a printing error. [*Actually the student provided the code as is. David*]

I will take a guess that the problem set is to perform multiplication using binary arithmetic by adding in 2n x the second number for each bit set in the first. E.g. 23*5 is 10111 x 510 = 80+0+20+10+5.

The algorithm you have chosen is to shift the first number right bit by bit, testing the least significant (right-most) bit, while at the same time left-shifting the second number to obtain the powers of 2. This is almost correct, but the first value of `num2` you are adding to the total has already been shifted, whereas it can be seen from the example above, the first addition should be the original entered value (5). This is what is meant by "…not including the first two numbers…". If you move the add-to-total to the beginning of the loop, then the total will be accumulated as 5+10+20…, which is what we want. We have now fixed the start-up conditions, so let's check the exit condition. We want to include all the bits from `num1`, so we want to continue the loop while there are still bits to process, in other words, while `num1` is not all zero bits. The current test `(num1!=1)` does not do this, it will stop with the last bit still not processed. Let's change the condition to `(num!=0)` and give the program a test:

```
Enter two numbers: 23 5
   23    5
   11   10
    5   20
    2   40
    1   80
the total is 115
```

We now have a working program, but the code does not quite capture the intent: it is required to demonstrate how binary multiply can be performed by shift and add operations, but there are no shifts! Although we know that in C and C++ `n/2` is equivalent to `n>>1` and `n*2` is equivalent to `n<<1`, using the bit shift operators would illustrate the intent more clearly. Similarly testing the least significant bit would be better as a mask and test rather than the remainder of a divide. Finally, the source code would benefit from some comments to explain what it is for.

Thus we end up with:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

// Perform binary multiply of two integers
// using shift and add.  The total is built up
// by adding the matching power of two from
// the second number for each bit set in the
// first bit. Bits are tested in the first
// number by testing the LS bit and shifting
// right. The powers of two of the second
// number are generated by shifting left on
// each iteration.
// e.g. 23(10111)*5 = 80 + 0 + 20 + 10 + 5
// (summed in reverse order).

int main() {
  while(1) {
    int num1, num2, total=0;
    cout << "Enter two numbers: ";
    cin >> num1 >> num2;
    while(num1 != 0) {
      cout << setw(5) << num1
           << setw(5) << num2 << endl;
      if((num1&0x01) != 0)
        total += num2;
      num1 >>= 1;
      num2 <<= 1;
    }
    cout << "the total is " << total << endl;
  }
  return 0;
}
```

The Winner of SCC 28

The editor's choice is:

Tony Houghton

Please email `francis@robinton.demon.co.uk` to arrange for your prize.

Francis' Commentary

For my comments on the first little program for SCC28 see my Francis' Scribbles column elsewhere (working late and meeting deadlines resulted in my sending David some code I had already used in my column. Fortunately that does not matter too much.)

Here is my commentary on the second little program.

```
#include<iostream>
#include<iomanip>
#include<string>

using namespace std;

int main() {
  while(1) {
    int num1, num2, total = 0;
    cout << "Enter two numbers: ";
    cin >> num1 >> num2;
    while (num1 != 1) {
      cout << setw(5) << num1
           << setw(5) << num2 << endl;
      num1 /= 2;
      num2 *= 2;
      if((num1 % 2) != 0) total += num2;
    }
    cout << "the total is " << total << endl;
  }  // End While 1
  return 0;
}
```

First, in the above restatement of the program I have reformatted the code to make its structure more visible. Now let me focus on that structure before turning to the root of the problem.

Prepare For Exceptions

I strongly advocate that the default form of the definition of `main()` should encapsulate its code in a `try` block. That seems to me to be a good discipline for students as soon as they are conscious that errors may occur at runtime that result in an exception. Indeed they should be encouraged to validate such things as input and exceptions make it easy for them to have a default action when validation fails. So I would start with:

```
int main() {
  try {
    // main code
  }
  catch(...) {
    cerr << "An exception occurred.\n";
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

Now that is a fixed framework and it isn't even tedious to type because you just copy and paste it from a text file of standard bits of source code.

Repetition of Process

The first problem with the student's code is that it has no termination. He puts it all in a forever loop without any internal exit. I have no problem with forever loops (except that I use `while(true)`) as long as they have an internal exit (that is unless you are writing a process which must continue until the machine is switched off). In this case I would write something such as:

```
while(true) {
  // main process
  cout << "Do you want another? (y/n)";
  char yn;
  cin >> yn;
  if(yn == 'n' || yn == 'N') break;
}
```

Now we can argue about the details but the general idea should be considered as an idiom of programming (not just C++).

Appropriate Variable Names

As an ex-maths teacher I quickly recognised the algorithm being implemented by this program (goes under several names, try using Google to search for 'Russian Peasant Multiplication') but the variable names were not helpful. Let me suggest some others and polish up the code whilst I am at it.

```
cout << "Enter the pair of numbers you want "
     << "to multiply together: ";
int multiplicand(getint());
int multiplier(getint());
int product(0);
```

Now the variable names give a hint as to what is being done. Validation and handling bad input is all handled by the `getint()` function. Actually I would use my `read()` set of templates but that is just a convenience. However avoiding repetitious coding whilst always validating input should be learnt from the very beginning.

Getting the Right Test

Now we get to the actual reason the program does not work. Look at that inner `while` loop. Look at the test. Surely this is too early, or it is the wrong test because it fails immediately if the multiplicand is one. In other words multiplying one by anything will give a product of zero. There are several solutions but I would prefer:

```
while(multiplicand)
```

or if you do not like that form:

```
while(multiplicand != 0)
```

Getting the Right Order of Statements

For the algorithm (which is effectively using binary for multiplication) we add in the current value of the multiplier into the product if the current value of the multiplicand is odd (i.e. would end in one in binary), then the multiplicand is halved (shifted left) and the multiplier is doubled (shifted right). The order of these operations is vital and is the second point of error in the student's code. Leaving aside the display of the interim results, which I would have preferred to see include the interim value of the product in a third column) the meat of the algorithm is coded as:

```
if(multiplicand % 2) product += multiplier;
multiplicand /= 2;
multiplier *= 2;
```

Putting It All Together

Here is my complete program (well I have left out the front matter of headers and using directive):

```
int main() {
  try {
    while(true) {
      cout << "Enter the pair of numbers you "
           << "want to multiply together: ";
      int multiplicand(getint());
      int multiplier(getint());
      int product(0);
      while(multiplicand) {
        cout << setw(5) << multiplicand
             << setw(5) << multiplier
             << setw(10) << product
             << '\n';
        if(multiplicand % 2)
          product += multiplier;
        multiplicand /= 2;
        multiplier *= 2;
      }
      cout << "The product is " << product
           << ".\n\n";
      cout << "Do you want another? (y/n)";
      char yn;
      cin >> yn;
      if(yn == 'n' || yn == 'N') break;
    }
  }
  catch(...) {
    cerr << "An exception occurred.\n";}
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

Note that the depth of nesting for structures suggests that some refactoring into functions might be desirable. I would probably make the outermost loop a `do-while` one and use the return value from a function asking about repeating the process in the `while`. Something like:

```
do {
  // process
while(do_again());
```

I would also move out the display of the intermediate results to a function. I would also probably replace the computation block with a function call, but that is much more marginal because all the arguments have to be passed by reference and I doubt that we get much more clarity in exchange for using a function.

Now do feel free to comment on the coding style and anything else that you want to criticise. Remember that this column is supposed to be a kind of seminar and not a lecture.

Student Code Critique 29

(Submissions to `scc@accu.org` by September 10th)

Looks like an ordinary snippet, doesn't it? Amazingly, it contains various mistakes for such a few lines. Please provide a correct version.

```
#include <iostream>
using std::cout;
using std::endl;

#include <list>
using std::list;

int main() {
  list<double>::iterator it;
    list<double> lst;
  *it = 34;
  *++it = 45;
    *++it = 87;
    it = lst.begin();
    for (;it < lst.end(); ++it){
      cout << it << '\t' << *it << endl;
  }
    system("pause");
  return 0;
}
```

Features

Professionalism in Programming #27

And Now For Something Completely Different...

Pete Goodliffe <pete@cthree.org>

Is it that time already? This is the 27th column in the professionalism series. It's been running for 4½ years now, in which time it's established itself as a part of the furniture. You either take it for granted and have learnt to carefully ignore it, or you diligently read and inwardly digest every word. Well, now it's time to break with tradition, and also to see how much attention you've been paying all these years.

The series index beside this article provides an overview of the secluded tributaries we've explored in the vast software development waterways. In this article we'll revisit some of this ground in a more interactive manner. It's your chance to see how 'professional' you really are.

Here's the game plan: for a few of these past topics I have posed some thought-provoking questions for you to mull over. Some are purely factual, some are more personal, probing your personal development practices and those of the team you work in. Consider them and answer as fully and honestly as you can. Then turn to the end of the article, where I provide my musings on each question. I won't be so bold as to claim this is a definitive answer set, but more of a gentle exploration of each problem.

You'll only get out of this exercise as much as you're prepared to put in. So grab a cup of coffee, find a comfy chair in a quiet corner, and it's eyes down for a full house…

Questions

First, here are the questions. Spend a while considering your answer to each one before you move on to the following section.

Defensive Programming (C Vu 13.5, August 2001)

In this article we looked at the need for a 'defensive' approach to programming, learnt to assume nothing, and investigated some practical defensive coding techniques. We saw how to use constraints effectively, as typified by C's `assert` macro. However:

1. Could there be such a thing as *too much* defensive programming?
2. Should assertions conditionally compile away to nothing in production builds? If not, which assertions should remain in release builds?
3. Should the defensive checking of pre- and postconditions be put *inside* each function, or beside each important function *call*?
4. Are constraints a perfect defensive tool? What are their drawbacks?
5. Can you *avoid* defensive programming?
   a) If you designed a *better* language, would defensive programming still be necessary? How could you do this?
   b) Does this show that C and C++ are flawed because they have so many areas for problems to manifest?
6. What sort of code need you not worry about writing defensively?
7. When you document a function, do you state the pre- and postconditions?
   a) Are they always implicit in the description of what it does?
   b) If there are no pre/postconditions do you explicitly document this?
8. Many companies pay lip service to defensive programming. Does your team recommend it? Take a look at the code base – do they really? How widely are constraints codified in assertions? How thorough is the error checking in each function?
9. Are you naturally paranoid enough? Do you look both ways

before crossing the road? Do you eat your greens? Do you check for every potential error in your code, no matter how unlikely?
   a) How easy is it to do this thoroughly? Do you forget to think about errors?
   b) Are there any ways to help yourself write more thorough defensive code?

Layout of Source Code (C Vu 12.2, April 2000)

This first ever professionalism column looked into the contentious topic of source code layout, demonstrating that consistency is more important than any one particular coding style. We concluded that holy wars over topics like this are pointless and unprofessional. So:

1. Should you alter the layout of legacy code to conform to your latest code style? Is this a valuable use of code reformatting tools?
2. One common layout convention is to split source lines at a set number of columns. What are the pros and cons of this approach?

Professionalism in Programming Index

This is a comprehensive catalogue of the professionalism series to date.

No	C Vu	Title/description	Date
1	12.2	Layout of Source Code	April 2000
2	12.3	Team Work	June 2000
3	12.4	Being Specific	August 2000
		Writing software specifications	
4	12.5	Code Reviews	October 2000
5	12.6	Documenting Code	December 2000
6	13.1	Good Design	February 2001
7	13.2	Practising Safe Source	April 2001
		Source control systems	
8	13.3	The Programmer's Toolbox	June 2001
9	13.4	Software Testing	October 2001
10	13.5	Defensive Programming	August 2001
11	13.6	Software Development: Fantasy, Fiction or Face	December 2001
		A cautionary parable for software developers	
12	14.1	Recipe For a Program	February 2002
		Software development methodologies	
13	14.2	How Long is a Piece of String?	April 2002
		Software time-scale estimation	
14	14.3	There and Back Again	June 2002
		Personal development	
15	14.4	The Outer Limits	August 2002
		Overview of programming disciplines	
16	14.5	What's in a Name?	October 2002
		Naming program elements appropriately	
17	14.6	The Code that Jack Built	December 2002
		Code build systems	
18	15.1	Engineering a Release	February 2003
		The real software development process	
19	15.2	A Passing Comment	April 2003
		Writing effective code comments	
20	15.3	Software Evolution or Software Revolution?	June 2003
		How software grows over time	
21	15.4	Software Architecture	August 2003
22	15.5	Finding Fault	October 2003
		Debugging your programs	
23	15.6	To Err is Human	December 2003
		Managing error conditions	
24	16.1	The Need For Speed (Part One)	February 2004
		Optimisation series	
25	16.2	The Need For Speed (Part Two)	April 2004
26	16.3	The Need For Speed (Part Three)	June 2004
27	16.4	And Now For Something Completely Different...	August 2004

3. How detailed should a *reasonable* coding standard be?
   a) How serious are deviations from the style? How many limbs should be amputated for not following it?
   b) Can such a specification become too detailed and restrictive? What would happen if it did?
4. Is good code *presentation* or good code *design* more important? Why?
5. Do you write in a consistent style?
   a) When you touch other people's code, what layout style do you adopt – theirs or your own?
   b) How much of your coding style is dictated by your editor's auto-formatting? Is this an adequate reason for adopting a particular style?
6. Tabs: are they a work of the devil, or the best thing since sliced bread? Explain why.
   a) Do you know if your editor inserts tabs automatically? Do you know what your editor's tab stop is?
   b) Some *hugely* popular editors indent with a mixture of tabs and spaces. Does this make the code any less maintainable?
   c) How many spaces should a tab correspond to?
7. Grab a text editor and have a go at this bit of C++, it calculates the n^{th} prime number. It's written in one particular coding style. Have a crack at presenting it as *you'd* like to see it. Don't try to change the implementation at all.

```
/* Returns whether num is prime. */
bool
isPrime(int num ) {
    for ( int x = 2; x &lt; num; ++x ) {
      if ( !(num % x ) ) return false;
    }
    return true;
}

/* This function calculates the 'n'th prime
   number.*/
int
prime( int pos ) {
    if ( pos ) {
        int x = prime( pos-1 ) + 1;
        while ( !isPrime( x ) ) {
            ++x;
        }
        return x;
    } else {
        return 1;
    }
}
```

What's in a Name? (C Vu 14.5, October 2002)

This article showed the impact of good naming on the quality of our source code, and demonstrated practical naming techniques for many common code constructs. What do you think about these issues:

1. Are these good variable names? Answer with either *yes* (explain why, and in what context), *no* (explain why), or *can't tell* (explain why).
   a) `int num_apples`
   b) `char foo`
   c) `bool num_apples`
   d) `char *string`
   e) `int loop_counter`
2. When would these be appropriate function names? What return types/parameters might you expect? What return types would make them nonsensical?
   a) `doIt(...)`
   b) `value(...)`
   c) `sponge(...)`
   d) `isApple(...)`
3. Should a naming scheme favour the easy reading or easy writing of code? How would you make either easy?
4. What do you do when naming conventions collide? Say you're working on camelCase C++ code, and need to do STL (using_underscore) library work. What's the best way to handle this situation?
5. If `assert` is a macro why is its name lower case? Why should we name macros so they stand out?
6. Long calculations can be made more readable by putting intermediate results in temporary variables. Suggest good naming heuristics for these types of variable.
7. Do you have to port code between platforms? How has this affected filenames, any other naming, and the overall code structure?

A Passing Comment (C Vu 15.2, April 2003)

The final article we'll exhume looked at how to write code comments. We learnt what makes a good comment, how to avoid pointless and intrusive comments, and how to use comments to help us write source code. Given this, then:

1. How might the *need for* and the *content of* comments differ in the following types of code:
   a) Low level assembly language (machine code)
   b) Shell scripts
   c) A single file test harness
   d) A large C/C++ project
2. You can run tools to calculate what percentage of your source code lines are comments. How useful are they? How accurate a measure is this of comment quality?
3. When you document a C/C++ API with a code comment block, should it go in the public header file that declares the function, or the source file containing the implementation? What are the pros and cons of each location?
4. Look carefully at the source files you've recently worked on. Inspect your commenting. Is it honestly any good? (I bet as you read through the code you'll find yourself making a few changes!)
5. How do you ensure that your comments are genuinely valuable and not just personal ramblings that only you can understand?
6. Do the people you work with all comment to the same standard, in about the same way?
   a) Who's the best at writing comments? Why do you think that? Who's the worst? How much of a correlation does this bear to their general quality of coding?
   b) Do you think any imposed coding standards could raise the quality of the comments written by your team?
7. Do you include history logging information in each source file? If yes:
   a) Do you do maintain it manually? Why, if your revision control system will insert this for you automatically? Is the history kept particularly accurate?
   b) Is this a *really* sensible practice? How often is this information needed? Why is it better placed in the source file than in another, separate mechanism?
8. Do you add your initials to or otherwise mark the comments you make in other people's code? Do you ever date comments? When and why do you do this – is it a useful practice? Has it ever been useful to find someone else's initials and time stamping?

Discussion and Answers

Lazy readers will have jumped here already. Please do spend some time considering your answer to each question first. It will be interesting to compare your response with mine. Do you disagree with anything? Do you agree? Let me know.

Defensive Programming

1. Could there be such a thing as *too much* defensive programming?

Yes – just as too many comments can degrade code readability, so too can many defensive checks. Redundant checks can be avoided with careful coding, for example with a good choice of types.

2. Should assertions conditionally compile away to nothing in production builds? If not, which assertions should remain in release builds?

People hold passionate beliefs on this subject. I don't think the answer is necessarily black and white; I can see the argument both ways. There are always some very nit-picky assertions that really won't *need* to be left in production builds. But some assertion occurrences may still interest you in the field. Now, if you do leave any constraint checks in, they *must* change behaviour – the program shouldn't abort on failure, just log the problem and move on.

Remember, real run-time error checks should *never* be removed; they should never be coded in assertions anyway.

3. Should the defensive checking of pre- and postconditions be put *inside* each function, or beside each important function *call*?

In the function, without a doubt. This way, you only need to write tests once. The only reason you'd want to move them out is to gain flexibility, to choose what happens when a constraint fails. This isn't a compelling gain for such an explosion in complexity and potential for failure.

4. Are constraints a perfect defensive tool? What are their drawbacks?

No, they are nowhere near perfect. Redundant constraints can be at best a pest, and at worst a hindrance. For example, you could `assert` that a function parameter `i >= 0`. But it's much better to make `i` an unsigned type that can't contain 'invalid values' anyway.

Treat constraints that can be compiled out with a certain degree of suspicion: we must carefully check for any side effects (assertions can have subtle indirect consequences), and for timing issues in the debug build that alters its behaviour from a release build. Ensure that assertions are logical constraints and not genuine run-time checks that mustn't be compiled out. It *is* possible to put bugs in the bug-defence code!

Carefully used, constraints are still far better than dancing barefoot over the hot coals of chance.

5. Can you *avoid* defensive programming?

a) If you designed a *better* language, would defensive programming still be necessary? How could you do this?

b) Does this show that C and C++ are flawed because they have so many areas for problems to manifest?

Some language features certainly could be designed to avoid errors. For example, C doesn't check the index of any array lookup you perform. As a result you can crash the program by accessing an invalid memory address. The Java runtime, on the other hand, checks *every* array index before lookup, so such an catastrophe will never arise. (Bad indexes will still cause an error though, which is just a better defined class of failure).

Despite the long list of 'improvements' you could make to the liberal C specification (and I urge you to think of as many as you can) you'll never be able to create a language that doesn't need defensive programming. Functions will always need to validate parameters, and classes will always need invariants to check their data is internally consistent.

Although C and C++ do provide plenty of opportunity for things to go wrong, they also provide a great deal of power and expression. Whether that makes the languages 'flawed' depends on your viewpoint – this is a topic ripe for holy war.

6. What sort of code need you not worry about writing defensively?

I've worked with people who refused to put any defensive code into an old program because it was *so bad* that their defences would make no difference. I managed to resist the urge to whack them with a large mallet!

You might argue that a small, stand-alone, single file program or perhaps a small test harness file doesn't need this sort of careful defensive code or any rigorous constraints – but even in these situations not being careful is just being sloppy. We should be defensive all the time.

7. When you document a function, do you state the pre- and postconditions?

a) Are they always implicit in the description of what it does?

b) If there are no pre/postconditions do you explicitly document this?

No matter how obvious you think a contract is from the function name or its description, explicitly stating the constraints removes any ambiguity – remember, it's always better to remove areas of assumption. Explicitly writing *Preconditions: None* will document a contract explicitly.

8. Many companies pay lip service to defensive programming. Does your team recommend it? Take a look at the codebase – do they really? How widely are constraints codified in assertions? How thorough is the error checking in each function?

Very few companies have a culture of excellent code with the right level of defence. Code reviews are a good way to bring a team's code up to a reasonable standard; many eyes see many more potential errors.

9. Are you naturally paranoid enough? Do you look both ways before crossing the road? Do you eat your greens? Do you check for every potential error in your code, no matter how unlikely?

a) How easy is it to do this thoroughly? Do you forget to think about errors?

b) Are there any ways to help yourself write more thorough defensive code?

No one finds it naturally easy – thinking the worst of your carefully crafted new code runs contrary to the programmer instinct. Instead, expect the worst of any people who will be using your code. They're nowhere near as conscientious a programmer as you!

A very helpful technique is to write unit tests for each function/class. Some experts strongly advise doing this *before* writing a function; this makes a lot of sense. It helps you to think about all the error cases, rather than blithely trusting that your code will work.

Layout of Source Code

1. Should you alter the layout of legacy code to conform to your latest code style? Is this a valuable use of code reformatting tools?

It's usually safest to leave legacy code however you find it, even if it's ugly and hard to work with. I'd only entertain reformatting if I was absolutely sure that none of the original authors would ever need to return.

By reformatting you lose the ability to easily compare a particular revision of the source with a previous one – you'll be thrown by many, many, formatting changes which may hide the one difference you really needed to see. You also risk introducing program errors in the reformatting.

As far as code reformatting tools go, they're nice curiosities, but I don't advocate the use of them. Some companies insist on running source files through these tools before checking any code into their repository. The advantage is that all source is homogenised, pasteurised, and uniformly formatted. The major disadvantage is that no tool is perfect; you'll lose some helpful nuances of the author's layout. Unless all the programmers on your team are gibbons, don't use a reformatting tool.

2. One common layout convention is to split source lines at a set number of columns. What are the pros and cons of this approach?

As with many such presentation concerns there is no absolute answer; it is a matter of personal taste.

I like to split my code up so that it fits on an 80 column display. I've always done that, so it's a matter of habit as much as anything else. I don't disagree with people who like long lines, but I find long lines hard to work with. I set my editor up to 'wrap' continuous lines rather than provide a horizontal scrollbar (horizontal scrolling is clumsy). In this environment long lines tend to ruin the effect of any indentation.

As I see it, the main advantage of fixed column widths is not printability, as some would claim. It's the ability to have several editor windows open side-by-side on the same display.

In practice, C++ seems to produce very long lines. It's more verbose than C; you end up calling member functions on objects referenced by another object through a templated container... There are other strategies to manage the many, many, long lines this may lead to. You could store intermediate references in temporary variables, for example.

3. How detailed should a *reasonable* coding standard be?

a) How serious are deviations from the style? How many limbs should be amputated for not following it?

b) Can such a specification become too detailed and restrictive? What would happen if it did?

Six limbs should be amputated for deviations from any coding standard.

I have seen many coding standards that are so prescriptive and paralysing that the poor programmers have just plain ignored it. To be useful, and to be accepted, a coding standard should provide a little room for manoeuvre, perhaps with a *best practice* approach given as an example.

4. Is good code *presentation* or good code *design* more important? Why?

This is really a very artificial question. Both are fundamental for good code, and you should never be asked to sacrifice one for the other. If you ever are, fear. However, which one you just chose may say a lot about you as a programmer.

Bad formatting is certainly easier to fix than bad design, especially if you use clever tools to homogenise your code's formatting.

There is an interesting connection between presentation and design: Bad presentation often shows that the code was produced by a bad programmer, which probably means that it suffers from bad internal design too. Or it may imply that the code has been maintained by a series of different programmers, with a subsequent loss of the initial code design.

5. Do you write in a consistent style?

a) When you touch other people's code, what layout style do you adopt – theirs or your own?

b) How much of your coding style is dictated by your editor's auto-formatting? Is this an adequate reason for adopting a particular style?

If you can't alter the way your editor positions the cursor for you then you shouldn't be using it (either you're too inept, or your editor is).

If you can't write code in a consistent style then you should have your programmer's licence revoked. If you can't follow someone else's presentation style then you should be forced to maintain BASIC for the rest of your career.

[*FORTRAN 77 is far more restrictive on its code style and formatting. Depending on the flavour of BASIC, style and format can be varied greatly – Ed*]

6. Tabs: are they a work of the devil, or the best thing since sliced bread? Explain why.

a) Do you know if your editor inserts tabs automatically? Do you know what your editor's tabstop is?

b) Some *hugely* popular editors indent with a mixture of tabs and spaces. Does this make the code any less maintainable?

c) How many spaces should a tab correspond to?

Since this is such a religious issue, I'll just say *tabs: they suck* and back away quickly. Well, actually I'll add afterwards that the only thing more evil than indenting with tabs is indenting with tabs *and* spaces – nightmare!

If your editor *is* inserting tabs (and probably spaces) without you noticing, try using another editor for a while, to appreciate how frustrating it is. Try setting your tabstop to a different value and see what a mess it makes of the code. *Everyone uses the same editor, so it doesn't matter* is not a professional attitude. They don't.

You'll hear people recommend their choice of tabstop length and carefully justify their opinion. That's all very well; in fact a respected study claims that a *three* or *four* space tabstop provides optimum readability. (I favour four spaces because I don't like odd numbers!) However, a tab should 'correspond' to *no* fixed number of spaces. A tab is a tab, which is not a space or any multiple thereof. For code laid out using tabs, it shouldn't matter exactly how many spaces the tab is displayed as – the code should read well regardless. Unfortunately, I have rarely seen tab-indented code that works this way. All too often tabs and spaces are mixed together to make code line up neatly. This works fine with a tabstop set as the author intended. It makes an unholy mess with any other setting.

7. Grab a text editor and have a go at this bit of C++, it calculates the n^{th} prime number. It's written in one particular coding style. Have a crack at presenting it as *you'd* like to see it. Don't try to change the implementation at all.

This is a representative bit of Real World code, so don't dismiss this as a stupid and tedious exercise. Note that I haven't given any suggested answer here. My reformatting is just as valid as yours, and indeed as the original format.

If you're reading these answers without chewing over the questions at all, go on – have a go at this one. The magazine can wait whilst you type in a few lines...

Now, take a look at what you've written.

- How different is your version? How many specific changes did you make?
- For each change: is it a personal aesthetic preference, or can you justify the change with some rationale? Question this rationale – is it truly valid? How strongly would you be prepared to defend it?
- How comfortable were you with the original format? Did it bother you to read? Could you work in that coding style if you encountered code like it? *Should* you be able to become comfortable with it?

[*I have to admit I found it very difficult to restrain myself from reformatting the code as I processed this article! – Production Ed*]

What's in a Name?

1. Are these good variable names? Answer with either *yes* (explain why, and in what context), *no* (explain why), or *can't tell* (explain why).

a) `int num_apples`
b) `char foo`
c) `bool num_apples`
d) `char *string`
e) `int loop_counter`

The quality of a name depends on its context, and we can't honestly tell whether any of these are good or bad names. That's why the question asks for example contexts. There are some obvious contexts where the names might be bad: `num_apples` wouldn't be a particularly good name for a grapefruit counter.

`foo` is *never* a good name. I've yet to see anyone counting *foo*s. `loop_counter` is also bad; even if a loop gets too big for a short counter name you can still pick a more descriptive name, one that reflects the actual *use* of loop counter rather than its role *as* a loop counter.

We can't really tell whether `bool num_apples` is a good name, but it looks like it's not – a boolean cannot hold a number. Perhaps it's recording whether a separate count of apples is valid, but in this case it ought to be called something like `is_num_apples_valid`.

2. When would these be appropriate function names? What return types/parameters might you expect? What return types would make them nonsensical?

a) `doIt(...)`
b) `value(...)`
c) `sponge(...)`
d) `isApple(...)`

What each of these might mean depends on where you find them. Again we see how a name depends on its context, and that context can be provided by the enclosing scope of the function. Context information can even be given by function parameters or return variables.

3. Should a naming scheme favour the easy reading or easy writing of code? How would you make either easy?

How many times do you write a piece of code? (Think about it). How many times do you read it? That should give some indication as to the relative importances.

4. What do you do when naming conventions collide? Say you're working on camelCase C++ code, and need to do STL (using_underscore) library work. What's the best way to handle this situation?

I've worked on C++ codebases that used such a collision of naming conventions to their advantage. The internal logic used camelCase, whereas libraries and components that could be considered extensions of the standard library followed STL naming conventions. It actually worked quite well.

Unfortunately, it doesn't always work that nicely. I've seen plenty of inconsistent code where there was no rhyme or reason behind the changing styles.

5. If `assert` is a macro why is its name lower case? Why should we name macros so they stand out?

`assert` isn't capitalised because `assert` isn't capitalised. In an ideal world it would be, but standards being what they are we have to live with this tatty macro name. Sigh.

Macros and `#defined` constant definitions are downright dangerous – adopting the UPPER CASE name convention will prevent nasty collisions with ordinary names. It's as sensible as wearing glasses when a lunatic is walking round with a big pointy stick.

Because macros can be so painful you should chose names that are very unlikely to cause headaches. More importantly, avoid using the preprocessor as much as humanly possible.

6. Long calculations can be made more readable by putting intermediate results in temporary variables. Suggest good naming heuristics for these types of variable.

Bad temporary names are `tmp`, `tmp1`, `tmp2`... or `a`, `b`, `c`... These, unfortunately, are all common intermediate names.

Like any other item, temporary names should be meaningful. In fact, in a complex calculation good names can really serve to document the internal logic, showing what's going on.

If you find a value that really has no nameable purpose, if it truly is an arbitrary intermediate value that's hard to name, then you'll begin to understand why `tmp` is so popular. Avoid calling anything `tmp` if possible – try to break the calculation in some other way to help it make more sense.

7. Do you have to port code between platforms? How has this affected filenames, any other naming, and the overall code structure?

Older filing systems limited the number of characters you could use in a filename. This made file naming much messier. Unless you have to port code to such an archaic system this kind of limitation can be safely ignored.

File-based polymorphism is a cunning way to exploit filenames to achieve code substitutability at build-time. It's often used to select platform-specific implementations in portable code. You can set up header file search paths, allowing one `#include` to pull in a different file depending on the current build platform.

A Passing Comment

1. How might the *need for* and the *content of* comments differ in the following types of code:

a) Low level assembly language (machine code)
b) Shell scripts
c) A single file test harness
d) A large C/C++ project

Assembly language is less expressive, providing fewer opportunities for self-documenting code. You'd therefore expect more comments in assembly code, and those comments to be at a much lower level than in other languages. Although the golden comment rule is "comments describe *why* not *how*", generally assembly comments *would* explain how as well as why.

There isn't an enormous a difference between the remaining three. Shell scripts can be quite hard to read back; they are a proto-Perl in this respect. Careful commenting helps. You're more likely to use literate programming techniques on a large C/C++ codebase.

2. You can run tools to calculate what percentage of your source code lines are comments. How useful are they? How accurate a measure is this of comment quality?

This kind of metric will give an insight into the code, but you shouldn't get too concerned about it. It isn't an accurate reflection of code quality. Well documented code might not contain *any* comments. Enormous revision histories or large corporate copyright messages can dominate small files, affecting this metric.

3. When you document a C/C++ API with a code comment block, should it go in the public header file that declares the function, or the source file containing the implementation? What are the pros and cons of each location?

This question was the cause of a big fight at one place I worked. Some argued for descriptions to go in the `.c` file. Being close to the function means that it's harder to write an incorrect comment, and harder to write code that doesn't match the documentation. The comment is also more likely to be changed in line with any code changes.

However, when placed in a header file, the description is visible alongside the public interface; a logical location. Why should someone have to look into the implementation to read any public API docs?

A *literate programming* documentation tool should be able to pull comments out of either place, but sometimes it's quicker not to use the tool and just read comments in the source; a bonus of the literate code approach. I favour placing the comments in header files.

Of course, in Java it's all one file anyway, and you'd conventionally use the Javadoc format.

4. Look carefully at the source files you've recently worked on. Inspect your commenting. Is it honestly any good? (I bet as you read through the code you'll find yourself making a few changes!)

When you read and review your own code it's very easy to skip the comments, presuming they're correct, or at least adequate. It is a good idea to spend some time looking at them, to assess how well they're written. Perhaps you could ask a trusted colleague to give you their (constructive) opinion on your commenting style.

5. How do you ensure that your comments are genuinely valuable and not just personal ramblings that only you can understand?

Some considerations for this are: write whole sentences, avoid abbreviations, keep comments neatly formatted, and in a common language (both the native language and the selection of words used from the problem domain). Avoid in-jokes, any throw-away statements, or anything that you're not entirely sure about.

Code reviews will highlight weaknesses in your comment strategy.

6. Do the people you work with all comment to the same standard, in about the same way?

a) Who's the best at writing comments? Why do you think that? Who's the worst? How much of a correlation does this bear to their general quality of coding?

b) Do you think any imposed coding standards could raise the quality of the comments written by your team?

Use code reviews to inspect your peers' comment quality, and to move your team towards a consistent quality of commenting.

7. Do you include history logging information in each source file? If yes:

a) Do you do maintain it manually? Why, if your revision control system will insert this for you automatically? Is the history kept particularly accurate?

b) Is this a *really* sensible practice? How often is this information needed? Why is it better placed in the source file than in another, separate mechanism?

It's human nature not to keep a history accurate, even with the best intentions in the world. It requires a lot of manual work that gets pushed out when time is tight. You should use tools to help, and put the right information in the right place (which I don't believe is the source file at all).

8. Do you add your initials to or otherwise mark the comments you make in other people's code? Do you ever date comments? When and why do you do this – is it a useful practice? Has it ever been useful to find someone else's initials and timestamping?

For some comments this is a useful practice. In other places, it's just inconvenient – extra comment noise that you have to read past to get to the really interesting stuff.

It's most useful with temporary `FIXME` or `TODO` comments, marking work in progress. Released production code probably shouldn't have these; no finished code should need a reader to understand the author or date of a particular change.

And Finally…

Just in case you got this far, here's a parting shot – one final question for a few bonus points. Answer this: How many programmers does it take to change a light bulb?

I have five answers. If you're good then I'll tell you next time. Let me know your answers.

Pete Goodliffe

[With apologies to the ghost of Monty Python for the second vague and fleeting reference of this series. Do you remember the first one?]

Creating Standard GUI Applications

Mark Summerfield and Jasmin Blanchette

In this second installment of our series on GUI programming with the Qt C++ toolkit, we're going to see how to create a standard GUI application, with a menu, toolbar, status bar, and a central area. The application is a simple image viewer that can display images in any of the formats that your installed version of Qt supports.

The `main()` function, in `main.cpp`, is straightforward:

```
#include <qapplication.h>
#include "viewer.h"

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  Viewer *viewer = new Viewer;
  viewer->show();
  viewer->connect(&app,
                  SIGNAL(lastWindowClosed()),
                  &app,
                  SLOT(quit()));
  return app.exec();
}
```

We create a `QApplication` object and then the form which we immediately show. The "signal–slot" connection ensures that when the last top-level window (in this case the only window) is closed, the application will terminate. Finally we start the event loop and wait for user interactions.

Qt provides the `QMainWindow` class as the base class for main windows, so we will subclass this and specialise it to our needs. Here's the definition which we've put in `viewer.h`:

```
#ifndef VIEWER_H
#define VIEWER_H

#include <qmainwindow.h>

class QAction;
class QLabel;

class Viewer : public QMainWindow {
  Q_OBJECT

public:
  Viewer(QWidget *parent = 0);

private slots:
  void openFile();

private:
  QAction *openFileAction;
  QAction *quitAction;

  QLabel *imageLabel;
  QString fileName;
};

#endif
```

We need the `Q_OBJECT` macro because we are using Qt's "signals and slots" mechanism in our subclass. We will connect the "open file" action (`File|Open`) to the `openFile()` slot. We'll explain the rest as we describe the implementation in `viewer.cpp`, piece by piece, starting with the includes.

```
#include <qaction.h>
#include <qfiledialog.h>
#include <qimage.h>
#include <qlabel.h>
#include <qmenubar.h>
#include <qpopupmenu.h>
#include <qstatusbar.h>
#include <qtoolbar.h>
#include "viewer.h"
#include "icon.xpm"
#include "openfile.xpm"
```

We include all the Qt classes we need, `viewer.h`, and also two XPM files. The XPM file format is an image format that is also valid C++. XPM images are easy to find on the Internet, and Linux distributions come with lots of them.

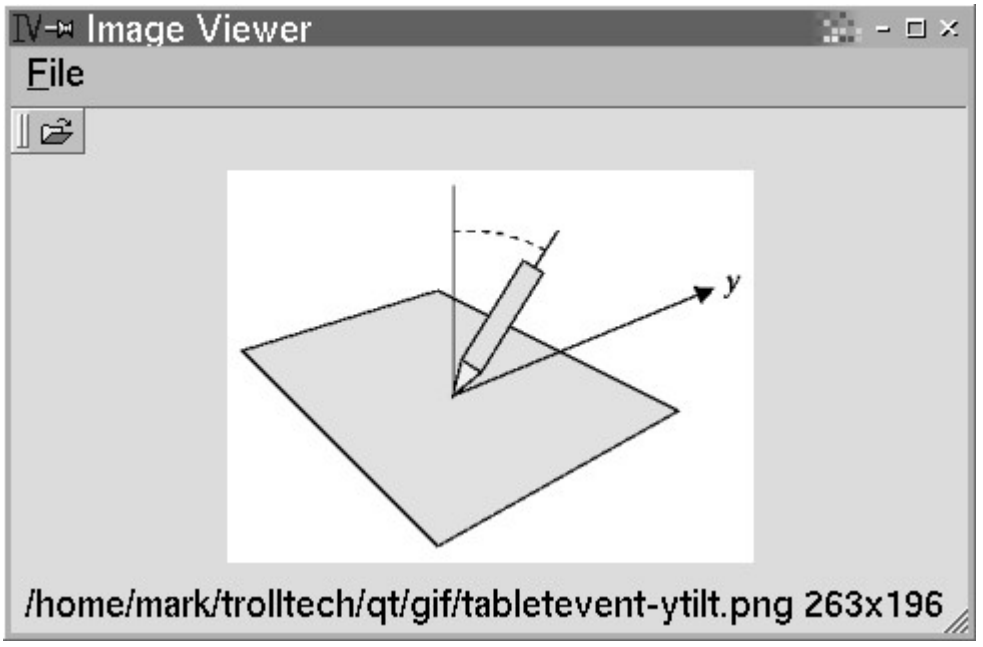
The constructor for a main window application is usually concerned with the creation of "actions": these are are "activated'' when the user clicks their associated menu option or toolbar button, or types their keyboard shortcut. Main windows usually have very simple layouts since they often contain either a single widget or an MDI (multiple document interface) workspace.

```
Viewer::Viewer(QWidget *parent)
               : QMainWindow(parent),
                 fileName(".") {
  imageLabel = new QLabel(this);
  imageLabel->setAlignment(AlignCenter);
  setCentralWidget(imageLabel);

  openFileAction =
        new QAction(QPixmap(openfile_xpm),
                    "&Open...",
                    CTRL+Key_O, this);
  connect(openFileAction,
          SIGNAL(activated()),
          this,
          SLOT(openFile()));
  quitAction = new QAction("&Quit",
                           CTRL+Key_Q,
                           this);
  connect(quitAction, SIGNAL(activated()),
          this, SLOT(close()));

  QPopupMenu *fileMenu = new QPopupMenu(this);
  openFileAction->addTo(fileMenu);
  quitAction->addTo(fileMenu);
  menuBar()->insertItem("&File", fileMenu);

  QToolBar *fileTools =
            new QToolBar("File Tools", this);
  openFileAction->addTo(fileTools);
```



```
    setCaption("Image Viewer");
    setIcon(QPixmap(viewer_xpm));

    statusBar()->message("Ready");
  }
```

We begin by creating a label widget that will be used to display the image. We then create two actions, the first "open file'', has an icon (`openfile.xpm`) and a shortcut of `Ctrl+O`. We connect the action to our custom `openFile()` slot (implemented shortly). The "quit" action is connected to the built-in `close()` slot. After the actions are created we want to give them a visual representation in the user interface. We create a new menu and add the "open file" action, and the "quit" action to it. We also create a toolbar and add the "open file" action to it. Qt automatically keeps menus and toolbars in sync.

Finally we set the application's caption and icon, and display "Ready" on the status bar. The first time we call `menuBar()` and `statusBar()`, Qt creates them; this ensures they are only created if they're actually used.

When the user invokes the "open file" action (by choosing `File|Open`, by clicking the "open" toolbar button, or by pressing `Ctrl+O`), the `openFile()` slot is called.

```
void Viewer::openFile() {
  QStringList formats =
                    QImage::inputFormatList();
  QString filters = "Images (*."
                    + formats.join(" *.")
                              .lower() + ")";
  QString newFileName =
      QFileDialog::getOpenFileName(fileName,
                                   filters, this);
  if(!newFileName.isEmpty()) {
    fileName = newFileName;
    QPixmap pixmap(fileName);
    imageLabel->setPixmap(pixmap);
    statusBar()->message(QString("%1 %2x%3")
                .arg(fileName)
                .arg(pixmap.width())
                .arg(pixmap.height()));
  }
}
```

Qt can provide a list of the image formats it can read, and we use this to create a file filter. For example, the filter might look like this, "Images (*.bmp *.gif *.jpeg *.pbm *.pgm *.png *.ppm *.xbm *.xpm)". We use one of `QFileDialog`'s static convenience functions to get an image file name, and unless the user clicks `Cancel` (in which case `getOpenFileName()` returns an empty string), we load the image and put some information in the status bar. If we didn't need to access the image after reading it we could have simply used:

```
imageLabel->setPixmap(QPixmap(fileName));
```

To build and run the application, save the files in a directory of their own (e.g. `viewer`), change to that directory, and run the following commands:

```
qmake -project
qmake viewer.pro
make
```

The first command creates a project file, the second creates a makefile based on the project file, and the third builds the application. If you use Visual Studio, use `nmake` instead of `make`.

In the previous article we saw how to create a dialog and lay out widgets inside it. Now we've seen how to create a main window application. In the next installment we'll combine this knowledge to create an application that can interact with the user through a dialog, and later on we'll see how to create custom widgets with any look and behaviour we want.

Mark Summerfield and Jasmin Blanchette

Using a Live Linux Distribution

Silas S. Brown

I think it was in C Vu that I first heard about Knoppix, `www.knoppix.org` – a "live" Linux distribution that will run entirely from the CD, needing no installation or configuration. It is very good for demonstrating Linux, or for using a Linux desktop to sort something out on a computer that otherwise runs Windows. It can also be used to install Linux quickly; you get a Debian system that runs a mixture of testing and unstable packages, and you can do package management as you see fit. (A while ago I met someone who wanted a free entry-level CAD package, and it was quicker for me to install Linux and find a suitable package from `www.debian.org` than it would have been to look for an appropriate piece of Windows software.)

What is even more useful is a document called "Knoppix Remastering HOWTO", available at `www.knoppix.net/docs/index.php/KnoppixRemasteringHowto` – this explains how to copy Knoppix into a spare directory on your existing Linux hard drive, `chroot` into it and alter it as you see fit, and then create a new CD image of the altered version. This is useful for a number of reasons. Firstly, if you need to experiment with recent, less stable packages but you don't want to upgrade your existing stable Linux environment, you can safely mix both distributions using this method (although if you are doing `chroot` from a different distribution, make sure to use something like `su` – so as to set the environment variables correctly). Secondly, you can make customised bootable CDs whenever you want to (I turned the process into a script to make this easier, although the script is rather specific to my system so I'll leave that as an exercise).

On slower computers it can take a few hours to generate the CD image, but it is worth it. It means you are able to take your exact customised environment to anyone else's PC and run it there, so long as it is able to boot the CD (or the special floppy disk that you can write). You may have problems persuading certain laptops to do this, but most computers "out there" will be OK with it. An alternative is to use VNC to access your desktop remotely, but that needs a good Internet connection; customised CDs do not. There are all kinds of uses for this. If you're familiar with Linux then it will save a lot of time in comparison with messing around with everybody's Windows setups.

I experimented using re-writeable CDs rather than ordinary CDRs. Rewriteables are slower (both in writing and reading) and have reduced capacity, and they will only work if you can get your PC to boot off the CD recorder (they are not readable on ordinary CD-ROMs). However they do save on resources when you're testing, because they can be re-used.

If you are running on a machine with less than 256Mb of RAM then you almost certainly want to make a swap file to make things run faster. Swap files can be made on the hard disk on any FAT partition, and any existing Linux swap partitions will be used automatically. You can also save a persistent home directory on the hard disk. (When you're mastering the Knoppix CD, it's worth knowing that you shouldn't rely on anything being in the home directory on startup. If you need something to be there by default then you should arrange a boot-up script to put it there, but please make the script do a test first because the user might be running a persistent home directory on the hard drive and doesn't want it to be restored to the default each time.)

There are many other versions of Knoppix that other people have re-mastered. Gnoppix (`www.gnoppix.org`) is interesting because it is based on the stable Debian distribution, rather than testing/unstable, although I find that testing/unstable is fine in the context of bootable CDs because these systems have comparatively short uptimes and are used as desktop machines, not secure Internet servers. I tried Morphix (`www.morphix.org`) which is supposed to be easier to customise, but it does have its problems (in the current version at the time of writing, any extra packages you add are copied to RAM when the CD loads, which could cause problems if you want a lot of packages and RAM is limited) – it's probably better to invest time in remastering Knoppix yourself. Another interesting variant is Oralux (`www.oralux.org`) which is designed for blind people and takes you into an Emacs desktop with software speech synthesis (go on, try it – everyone should have this experience). If you need something other than Debian, there are also some live CDs based on Red Hat and on BSD, but I haven't tried these as yet. Use Google and DistroWatch if you want to find them.

Silas S Brown

An Introduction to Programming with GTK+ and Glade in ISO C and ISO C++

Roger Leigh <rleigh@debian.org>

What is GTK+?

GTK+ is a *toolkit* used for writing graphical applications. Originally written for the X11 windowing system, it has now been ported to other systems, such as Microsoft Windows and the Apple Macintosh, and so may be used for cross-platform software development. GTK+ was written as a part of the *GNU Image Manipulation Program* (GIMP), but has long been a separate project, used by many other free software projects, one of the most notable being the *GNU Network Object Model Environment* (GNOME) Project.

GTK+ is written in C and, because of the ubiquity of the C language, *bindings* have been written to allow the development of GTK+ applications in many other languages. This short tutorial is intended as a simple introduction to writing GTK+ applications in C and C++, using the current 2.0/2.2 version of `libgtk`. It also covers the use of the Glade user interface designer for *rapid application development* (RAD).

It is assumed that the reader is familiar with C and C++ programming, and it would be helpful to work through the "Getting Started" chapter of the GTK+ tutorial before reading further. The `GTK+`, `Glib`, `libglade`, `Gtkmm` and `libglademm` API references will be useful while working through the examples.

I hope you find this tutorial informative.

Building the Example Code

Several working, commented examples accompany the tutorial. They are also available from `http://people.debian.org/~rleigh/gtk/ogcalc/`. To build them, type:

```
./configure
make
```

This will check for the required libraries and build the example code. Each program may then be run from within its subdirectory.

I have been asked on various occasions to write a tutorial to explain how the GNU autotools work. While this is not the aim of this tutorial, I have converted the build to use the autotools as a simple example of their use.

Legal Bit

GTK+ Basics

Objects

GTK+ is an *object-oriented* (OO) toolkit. I'm afraid that unless one is aware of the basic OO concepts (classes, class methods, inheritance, polymorphism), this tutorial (and GTK+ in general) will seem rather confusing. On my first attempt at learning GTK+, I didn't "get" it, but after I learnt C++, the concepts GTK+ is built on just "clicked", and I understood it quite quickly.

The C language does not natively support classes, and so GTK+ provides its own object/type system, **GObject**. GObject provides objects, inheritance, polymorphism, constructors, destructors and other facilities such as reference counting and signal emission and handling. Essentially, it provides C++ classes in C. The syntax differs a little from C++ though. As an example, the following C++

```
myclass c;
c.add(2);
```

would be written like this using GObject:

```
myclass *c = myclass_new();
myclass_add(c, 2);
```

The difference is due to the lack of a `this` pointer in the C language (since objects do not exist). This means that class methods require the object pointer passing as their first argument. This happens automatically in C++, but it needs doing "manually" in C.

Another difference is seen when dealing with polymorphic objects. All GTK+ widgets (the controls, such as buttons, checkboxes, labels, etc.) are derived from `GtkWidget`. That is to say, a `GtkButton` *is a* `GtkWidget`, which *is a* `GtkObject`, which *is a* `GObject`. In C++, one can call member functions from both the class and the classes it is derived from. With GTK+, the object needs explicit casting to the required type. For example

```
GtkButton mybutton;
mybutton.set_label("Cancel");
mybutton.show();
```

would be written as

```
GtkButton *mybutton = gtk_button_new();
gtk_button_set_label(mybutton, "Cancel");
gtk_widget_show(GTK_WIDGET(mybutton))
```

In this example, `set_label()` is a method of `GtkButton`, whilst `show()` is a method of `GtkWidget`, which requires an explicit cast. The `GTK_WIDGET()` cast is actually a form of *run-time type identification* (RTTI). This ensures that the objects are of the correct type when they are used.

Objects and C work well, but there are some issues, such as a lack of type-safety of callbacks and limited compile-time type checking. Using GObject, deriving new widgets is complex and error-prone. For these, and other, reasons, C++ may be a better language to use. `libsigc++` provides type-safe signal handling, and all of the GTK+ (and GLib, Pango et. al.) objects are available as standard C++ classes. Callbacks may also be class methods, which makes for cleaner code, since the class can contain object data without having to resort to passing in data as a function argument. These potential problems will become clearer in the next sections.

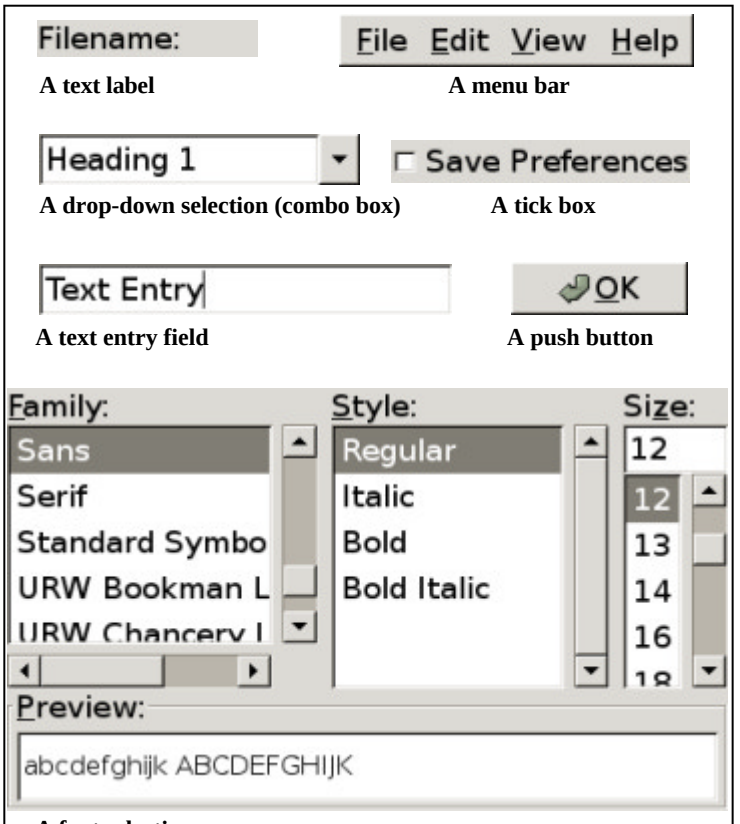


Figure 1: A selection of GTK+ widgets

Widgets

A user interface consists of different objects with which the user can interact. These include buttons which can be pushed, text entry fields, tick boxes, labels and more complex things such as menus, lists, multiple selections, colour and font pickers. Some example widgets are shown in Figure 1.

Not all widgets are interactive. For example, the user cannot usually interact with a label, or a framebox. Some widgets, such as containers, boxes and event boxes are not even visible to the user (there is more about this in the next section).

Different types of widget have their own unique *properties*. For example, a label widget contains the text it displays, and there are functions to get and set the label text. A checkbox may be ticked or not, and there are functions to get and set its state. An options menu has functions to set the valid options, and get the option the user has chosen.

Containers

The top-level of every GTK+ interface is the *window*. A window is what one might expect it to be: it has a title bar, borders (which may allow resizing), and it contains the rest of the interface.

In GTK+, a `GtkWindow` *is a* `GtkContainer`. In English, this means that the window is a widget that can contain another widget. More precisely, a `GtkContainer` can contain exactly **one** widget. This is usually quite confusing compared with the behaviour of other graphics toolkits, which allow one to place the controls on some sort of "form".

The fact that a `GtkWidget` can only contain one widget initially seems quite useless. After all, user interfaces usually consist of more than a single button. In GTK+, there are other kinds of `GtkContainer`. The most commonly used are horizontal boxes, vertical boxes, and tables. The structure of these containers is shown in Figure 2.

Figure 2 shows the containers as having equal size, but in a real interface, the containers resize themselves to fit the widgets they contain.

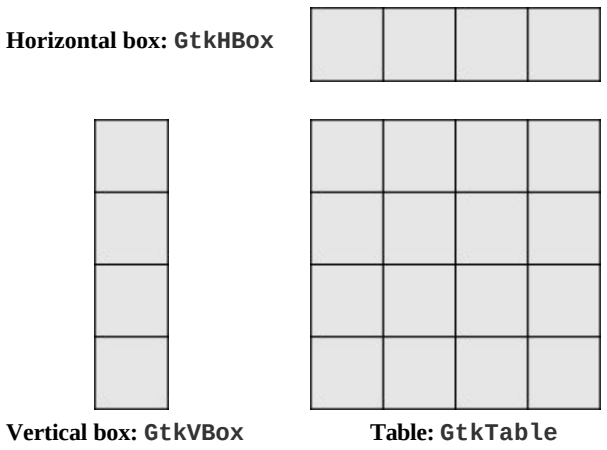


Figure 2: GTK+ containers
Each container may contain other widgets in the shaded areas. Containers may contain more containers, allowing them to nest. Complex interfaces may be constructed by nesting the different types of container.

In other cases, widgets may be expanded or shrunk to fit the space allotted to them. There are several ways to control this behaviour, to give fine control over the appearance of the interface.

In addition to the containers discussed above, there are more complex containers available, such are horizontal and vertical panes, tabbed notebooks, and viewports and scrolled windows. These are out of the scope of this tutorial, however.

Newcomers to GTK+ may find the concept of containers quite strange. Users of Microsoft Visual Basic or Visual C++ may be used to the free-form placement of controls. The placement of controls at fixed positions on a form has *no* advantages over automatic positioning and sizing. All decent modern toolkits use automatic positioning. This fixes several issues with fixed layouts:

- The hours spent laying out forms, particularly when maintaining existing code.
- Windows that are too big for the screen.
- Windows that are too small for the form they contain.
- Issues with spacing when accommodating translated text.
- Bad things happen when changing the font size from the default.

The nesting of containers results in a *widget tree*, which has many useful properties, some of which will be used use later. One important advantage is that they can dynamically resize and accommodate different lengths of text, important for internationalisation, when translations in different languages may vary widely in their size.

The Glade user interface designer can be very instructive when exploring how containers and widget packing work. It allows easy manipulation of the interface, and all of the standard GTK+ widgets are available. Modifying an existing interface is trivial, even when doing major reworking. Whole branches of the widget tree may be cut, copied and pasted at will, and a widget's properties may be manipulated using the "Properties" dialogue. While studying the code examples, Glade may be used to interactively build and manipulate the interface, to visually follow how the code is working. More detail about Glade is provided in a later section, where `libglade` is used to dynamically load a user interface.

Signals

Most graphical toolkits are *event-driven*, and GTK+ is no exception. Traditional console applications tend not to be event-driven; these programs follow a fixed path of execution. A typical program might do something along these lines:

- Prompt the user for some input
- Do some work
- Print the results

This type of program does not give the user any freedom to do things in a different order. Each of the above steps might be a single function (each of which might be split into helper functions, and so on).

GTK+ applications differ from this model. The programs must react to events, such as the user clicking on a button, or pressing Enter in an text entry field. These widgets emit signals in response to user actions. For each signal of interest, a function defined by the programmer is called. In these functions, the programmer can do whatever needed. For example, in the `ogcalc` program, when the "Calculate" button is pressed, a function is called to read the data from entry fields, do some calculations, and then display the results.

Each event causes a *signal* to be *emitted* from the widget handling the event. The signals are sent to *signal handlers*. A signal handler is a function which is called when the signal is emitted. The signal handler is *connected* to the signal. In C, these functions are known as *callbacks*. The process is illustrated graphically in Figure 3.

A signal may have zero, one or many signal handlers connected (registered) with it. If there is more than one signal handler, they are called in the order they were connected in.

Without signals, the user interface would display on the screen, but would not actually *do* anything. By associating signal handlers with

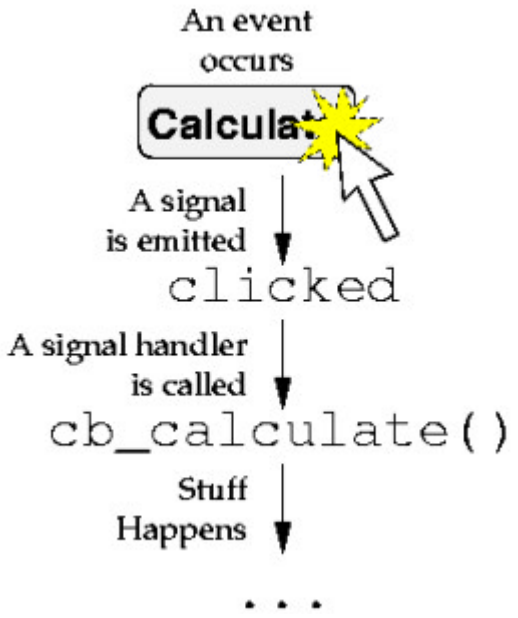


Figure 3: A typical signal handler.
When the button is pressed, a signal is emitted, causing the registered callback function to be called.

signals one is interested in, events triggered by the user interacting with the widgets will cause things to happen.

Libraries

GTK+ is comprised of several separate libraries:

`atk`	Accessibility Toolkit, to enable use by disabled people.
`gdk`	GIMP Drawing Kit (XLib abstraction layer – windowing system dependent part).
`gdk-pixbuf`	Image loading and display.
`glib`	Basic datatypes and common algorithms.
`gmodule`	Dynamic module loader (`libdl` portability wrapper).
`gobject`	Object/type system.
`gtk`	GIMP Tool Kit (windowing system independent part).
`pango`	Typeface layout and rendering.

When using `libglade` another library is required:

`glade`	User Interface description loader/constructor.

Lastly, when using C++, some additional C++ libraries are also needed:

`atkmm`	C++ ATK wrapper.
`gdkmm`	C++ GDK wrapper.
`gtkmm`	C++ GTK+ wrapper.
`glademm`	C++ Glade wrapper.
`pangomm`	C++ Pango wrapper.
`sigc++`	Advanced C++ signal/slot event handling (wraps GObject signals).

This looks quite intimidating! However, there is no need to worry, since compiling and linking programs is quite easy. Since the libraries are released together as a set, there are few library interdependency issues.

Designing an Application

Planning Ahead

Before starting to code, it is necessary to plan ahead by thinking about what the program will do, and how it should do it. When designing a graphical interface, one should pay attention to *how* the user will interact with it, to ensure that it is easy to understand, and efficient to use.

When designing a GTK+ application, it is useful to sketch the interface on paper, before constructing it. Interface designers such as Glade are helpful here, but a pen and paper are best for the initial design.

Introducing `ogcalc`

As part of the production (and quality control) processes in the brewing industry, it is necessary to determine the alcohol content of each batch at several stages during the brewing process. This is calculated using the density (gravity) in g/cm^3 and the refractive index. A correction factor is used to align the calculated value with that determined by distillation, which is the standard required by HM Customs & Excise. Because alcoholic beverages are only slightly denser than water, the PG value is (density-1) x 100. That is, 1.0052 would be entered as 52.

Original gravity is the density during fermentation. As alcohol is produced during fermentation, the density falls. Traditionally, this would be similar to the PG, but with modern high-gravity brewing (at a higher concentration) it tends to be higher. It is just as important that the OG is within the set limits of the specification for the product as the ABV.

The `ogcalc` program performs the following calculation:

$$O = (R \times 2.597) - (P \times 1.644) - 34.4165 + C$$

If O is less than 60, then

$$A = (O - P) \times 0.130$$

otherwise

$$A = (O - P) \times 0.134$$

The symbols have the following meanings:

A	Percentage Alcohol By Volume
C	Correction Factor
O	Original Gravity
P	Present Gravity
R	Refractive Index

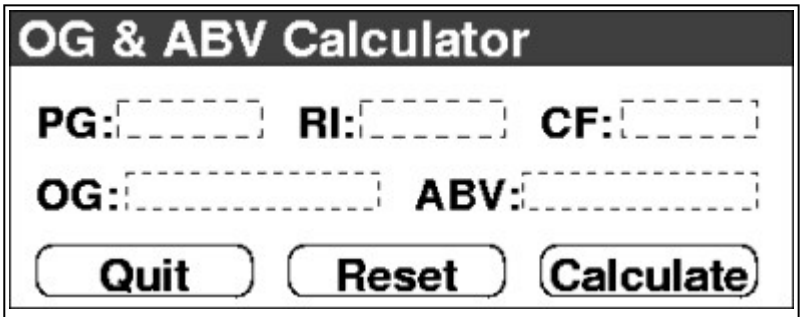


Figure 4: Sketching a user interface

The ogcalc main window is drawn simply, to illustrate its functionality. The top row contains three numeric entry fields, followed by two result fields on the middle row. The bottom row contains buttons to quit the program, reset the interface and do the calculation.

Designing the Interface

The program needs to ask the user for the values of `C`, `P`, and `R`. It must then display the results, `A` and `O`. A simple sketch of the interface is shown in Figure 4.

Creating the Interface

Due to the need to build up an interface from the bottom up, due to the containers being nested, the interface is constructed starting with the window, then the containers that fit in it. The widgets the user will use go in last. This is illustrated in Figure 5.

Once a widget has been created, signal handlers may be connected to its signals. After this is completed, the interface can be displayed, and the main *event loop* may be entered. The event loop receives events from the keyboard, mouse and other sources, and causes the widgets to emit signals. To end the program, the event loop must first be left.

GTK+ and C

Introduction

Many GTK+ applications are written in C alone. This section demonstrates the `C/plain/ogcalc` program discussed in the previous section. Figure 6 is a screenshot of the finished application.

This program consists of just three functions:

`on_button_clicked_reset()` – Reset the interface to its default state

`on_button_clicked_calculate` – Get the values the user has entered, do a calculation, then display the results.

`main()` – Initialise GTK+, construct the interface, connect the signal handlers, then enter the GTK+ event loop.

Code Listing

The program code is listed on pages 23-26. The source code is extensively commented, to explain what is going on.

To build the source, do the following:

```
cd C/plain
cc 'pkg-config –cflags gtk+-2.0' -c ogcalc.c
cc 'pkg-config –libs gtk+-2.0' -o ogcalc
ogcalc.o
```

Roger Leigh

[*I would recommend entering the code, it would be a valuable exercise in learning how a GTK application is built up. It may take a little while, but it certainly helped me to understand what is going on. – Ed*]

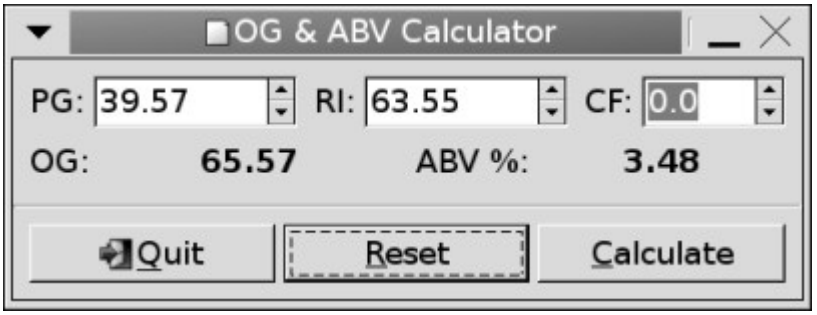
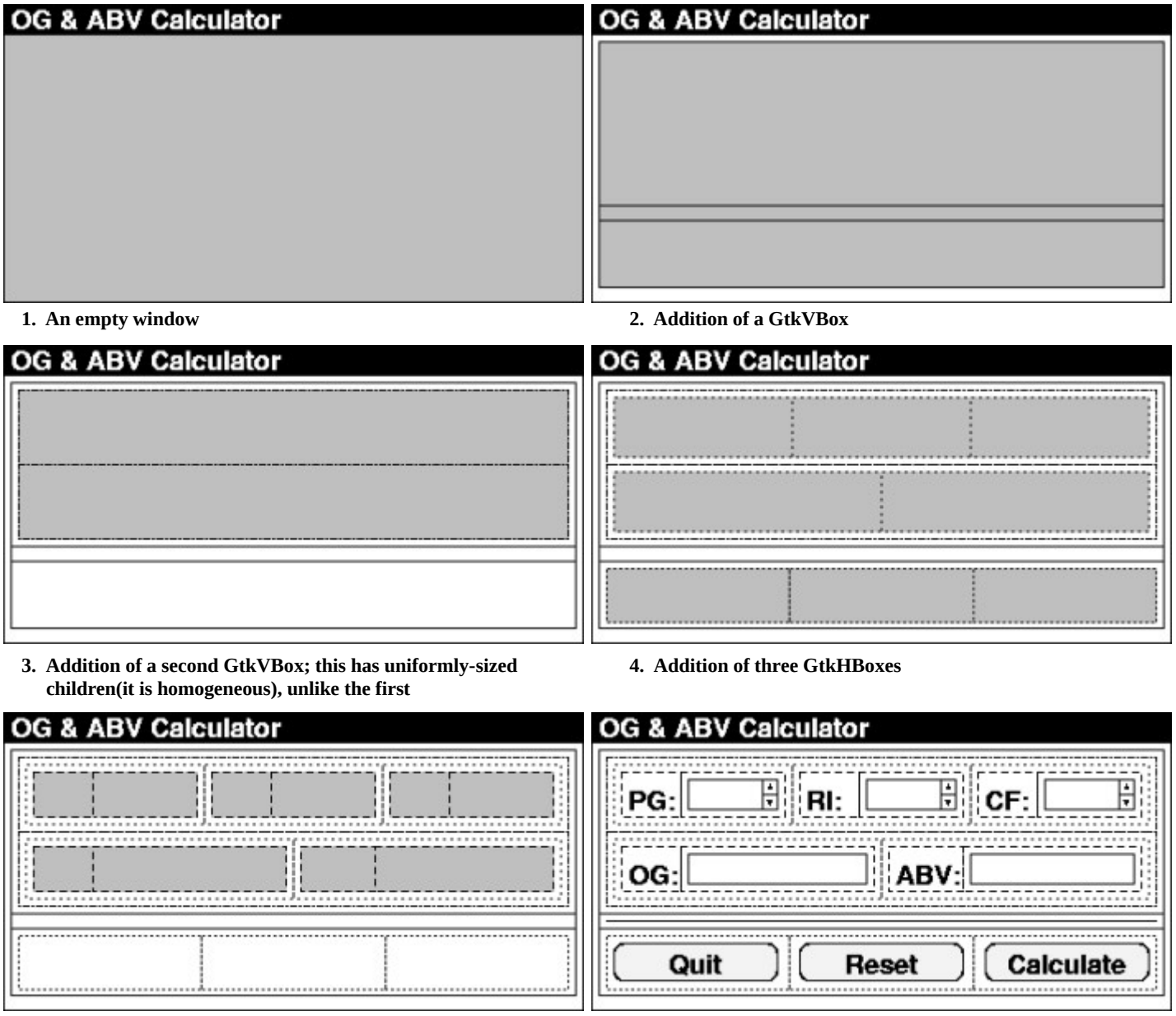


Figure 6: `C/plain/ogcalc` in action



1. An empty window
2. Addition of a GtkVBox
3. Addition of a second GtkVBox; this has uniformly-sized children(it is homogeneous), unlike the first
4. Addition of three GtkHBoxes
5. Addition of five more GtkHBoxes, used to ensure visually appealing widget placement
6. Addition of all of the user-visible widgets

Figure 5: Widget packing

The steps taken during the creation of an interface are shown, demonstrating the use of nested containers to pack widgets.

```
// C/plain/ogcalc.c
#include <gtk/gtk.h>

GtkWidget *create_spin_entry(
          const gchar *label_text,
          const gchar *tooltip_text,
          GtkWidget **spinbutton_pointer,
          GtkAdjustment *adjustment,
          guint digits);
GtkWidget *create_result_label(
          const gchar *label_text,
          const gchar *tooltip_text,
          GtkWidget **result_label_pointer);
void on_button_clicked_reset(
          GtkWidget *widget,
          gpointer data);
void on_button_clicked_calculate(
          GtkWidget *widget,
          gpointer data);

/* This structure holds all of the widgets
   needed to get all the values for the
   calculation. */
struct calculation_widgets {
  GtkWidget *pg_val;     // PG entry widget
  GtkWidget *ri_val;     // RI entry widget
  GtkWidget *cf_val;     // CF entry widget
  GtkWidget *og_result;  // OG result label
  GtkWidget *abv_result; // ABV% result label
};

int main(int argc, char *argv[]) {
  /* These are pointers to widgets used in
     constructing the interface, and later
     used by signal handlers. */
  GtkWidget *window;
  GtkWidget *vbox1, *vbox2, *hbox1, *hbox2,
            *button1, *button2;
  GtkObject *adjustment, *hsep;
  struct calculation_widgets cb_widgets;

  gtk_init(&argc, &argv); // Initialise GTK+.

  /* Create a new top-level window. */
  window =
         gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

```
/* Set the window title. */
gtk_window_set_title(GTK_WINDOW(window),
                     "OG & ABV Calculator");
/* Disable window resizing */
gtk_window_set_resizable(GTK_WINDOW(window),
                         FALSE);
/* Connect the window close button
   ("destroy" event) to gtk_main_quit(). */
g_signal_connect(G_OBJECT(window),
                 "destroy", gtk_main_quit,
                 NULL);

/* Create a GtkVBox to hold the other
   widgets.  This contains other widgets,
   which are packed in to it vertically. */
vbox1 = gtk_vbox_new(FALSE, 0);
/* Add the VBox to the Window.  A GtkWindow
  /is a/ GtkContainer which /is a/
  GtkWidget.  GTK_CONTAINER casts the
  GtkWidget to a GtkContainer, like a C++
  dynamic_cast. */
gtk_container_add(GTK_CONTAINER(window),
                  vbox1);
/* Display the Vbox.  At this point, the
   Window has not yet been displayed, so the
   window isn't yet visible. */
gtk_widget_show(vbox1);

/* Create a second GtkVBox.  Unlike the
   previous VBox, the widgets it will
   contain will be of uniform size and
   separated by a 5 pixel gap. */
vbox2 = gtk_vbox_new(TRUE, 5);
/* Set a 10 pixel border */
gtk_container_set_border_width(
                 GTK_CONTAINER(vbox2), 10);
/* Add this VBox to our first VBox. */
gtk_box_pack_start(GTK_BOX(vbox1), vbox2,
                   FALSE, FALSE, 0);

gtk_widget_show(vbox2);

/* Create a GtkHBox.  This is identical to a
   GtkVBox except that the widgets pack
   horizontally, instead of vertically. */
hbox1 = gtk_hbox_new(FALSE, 10);
/* Add to vbox2.  The function's other
   arguments mean to expand into any extra
   space alloted to it, to fill the extra
   space and to add 0 pixels of padding
   between it and its neighbour. */
gtk_box_pack_start(GTK_BOX(vbox2), hbox1,
                   TRUE, TRUE, 0);
gtk_widget_show (hbox1);

/* A GtkAdjustment is used to hold a numeric
   value: the initial value, minimum and
   maximum values, "step" and "page"
   increments and the "page size".  It's
   used by spin buttons, scrollbars, sliders
   etc. */
adjustment = gtk_adjustment_new(0.0, 0.0,
                     10000.0, 0.01, 1.0, 0);
/* Call a helper function to create a
   GtkSpinButton entry together with a label
   and a tooltip.  The spin button is stored
   in the cb_widgets.pg_val pointer for
   later use. */
hbox2 = create_spin_entry("PG:",
                "Present Gravity (density)",
                &cb_widgets.pg_val,
                GTK_ADJUSTMENT(adjustment),
                2);
/* Pack the returned GtkHBox into the
   interface. */
gtk_box_pack_start(GTK_BOX(hbox1), hbox2,
                   TRUE, TRUE, 0);
gtk_widget_show(hbox2);

/* Repeat the above for the next spin
   button. */
adjustment = gtk_adjustment_new (0.0, 0.0,
                     10000.0, 0.01, 1.0, 0);
hbox2 = create_spin_entry("RI:",
                "Refractive Index",
                &cb_widgets.ri_val,
                GTK_ADJUSTMENT(adjustment), 2);
gtk_box_pack_start(GTK_BOX(hbox1), hbox2,
                   TRUE, TRUE, 0);
gtk_widget_show(hbox2);

/* Repeat again for the last spin button. */
adjustment = gtk_adjustment_new (0.0, -50.0,
                          50.0, 0.1, 1.0, 0);
hbox2 = create_spin_entry("CF:",
                "Correction Factor",
                &cb_widgets.cf_val,
                GTK_ADJUSTMENT(adjustment),
                1);
gtk_box_pack_start(GTK_BOX(hbox1), hbox2,
                   TRUE, TRUE, 0);
gtk_widget_show(hbox2);

/* Now we move to the second "row" of the
   interface, for displaying the results. */

/* Firstly, a new GtkHBox to pack the labels
   into. */
hbox1 = gtk_hbox_new (TRUE, 10);
gtk_box_pack_start(GTK_BOX(vbox2), hbox1,
                   TRUE, TRUE, 0);
gtk_widget_show (hbox1);

/* Create the OG result label, then pack and
   display. */
hbox2 = create_result_label("OG:",
                "Original Gravity (density)",
                &cb_widgets.og_result);
gtk_box_pack_start(GTK_BOX(hbox1), hbox2,
                   TRUE, TRUE, 0);
gtk_widget_show(hbox2);

/* Repeat as above for the second result
   value. */
hbox2 = create_result_label("ABV %:",
                "Percent Alcohol By Volume",
                &cb_widgets.abv_result);
gtk_box_pack_start(GTK_BOX(hbox1), hbox2,
                   TRUE, TRUE, 0);
gtk_widget_show(hbox2);

/* Create a horizontal separator
   (GtkHSeparator) and add it to the VBox.*/
hsep = gtk_hseparator_new();
gtk_box_pack_start(GTK_BOX(vbox1), hsep,
                   FALSE, FALSE, 0);
gtk_widget_show(hsep);

/* Create a GtkHBox to hold the bottom row
   of buttons. */
hbox1 = gtk_hbox_new(TRUE, 5);
gtk_container_set_border_width(
                  GTK_CONTAINER(hbox1), 10);
gtk_box_pack_start(GTK_BOX(vbox1), hbox1,
                   TRUE, TRUE, 0);
gtk_widget_show(hbox1);
```

```
/* Create the "Quit" button.  We use a
   “stock” button—commonly-used buttons that
   have a set title and icon. */
button1 =
  gtk_button_new_from_stock(GTK_STOCK_QUIT);
/* We connect the "clicked" signal to the
   gtk_main_quit() callback which will end
   the program. */
g_signal_connect(G_OBJECT(button1),
           "clicked", gtk_main_quit, NULL);
gtk_box_pack_start(GTK_BOX(hbox1), button1,
                   TRUE, TRUE, 0);
gtk_widget_show(button1);

/* This button resets the interface. */
button1 =
     gtk_button_new_with_mnemonic("_Reset");
/* The "clicked" signal is connected to the
   on_button_clicked_reset() callback above,
   and our "cb_widgets" widget list is
   passed as the second argument, cast to a
   gpointer (void). */
g_signal_connect(G_OBJECT(button1),
        "clicked",
        G_CALLBACK(on_button_clicked_reset),
        (gpointer)&cb_widgets);
/* g_signal_connect_swapped is used to
   connect a signal from one widget to the
   handler of another.  The last argument is
   the widget that will be passed as the
   first argument of the callback.  This
   causes gtk_widget_grab_focus to switch
   the focus to the PG entry. */
g_signal_connect_swapped(G_OBJECT(button1),
   "clicked",
   G_CALLBACK(gtk_widget_grab_focus),
   (gpointer)GTK_WIDGET(cb_widgets.pg_val));
/* This lets the default action (Enter)
   activate this widget even when the focus
   is elsewhere. This doesn't set the
   default, it just makes it possible to
   set.*/
GTK_WIDGET_SET_FLAGS(button1,
                     GTK_CAN_DEFAULT);
gtk_box_pack_start(GTK_BOX(hbox1), button1,
                   TRUE, TRUE, 0);
gtk_widget_show(button1);

/* The final button is the Calculate button.*/
button2 =
 gtk_button_new_with_mnemonic("_Calculate");
/* When the button is clicked, call the
   on_button_clicked_calculate() function.
   This is the same as for the Reset button.*/
g_signal_connect(G_OBJECT(button2),
    "clicked",
    G_CALLBACK(on_button_clicked_calculate),
   (gpointer)&cb_widgets);
/* Switch the focus to the Reset button when
   the button is clicked. */
g_signal_connect_swapped(G_OBJECT(button2),
          "clicked",
          G_CALLBACK(gtk_widget_grab_focus),
          (gpointer)GTK_WIDGET(button1));
/* As before, the button can be the default.*/
GTK_WIDGET_SET_FLAGS(button2,
                     GTK_CAN_DEFAULT);
gtk_box_pack_start(GTK_BOX(hbox1),
                   button2, TRUE, TRUE, 0);
/* Make this button the default.  Note the
   thicker border in the interface—this
   button is activated if you press enter in
   the CF entry field. */
  gtk_widget_grab_default(button2);
  gtk_widget_show(button2);

  /* Set up data entry focus movement.  This
     makes the interface work correctly with
     the keyboard, so that you can touch-type
     through the interface with no mouse usage
     or tabbing between the fields. */

  /* When Enter is pressed in the PG entry
     box, focus is transferred to the RI
     entry. */
  g_signal_connect_swapped(
     G_OBJECT(cb_widgets.pg_val), "activate",
     G_CALLBACK(gtk_widget_grab_focus),
     (gpointer)GTK_WIDGET(cb_widgets.ri_val));
  /* RI -> CF. */
  g_signal_connect_swapped(
     G_OBJECT(cb_widgets.ri_val), "activate",
     G_CALLBACK(gtk_widget_grab_focus),
     (gpointer)GTK_WIDGET(cb_widgets.cf_val));
  /* When Enter is pressed in the RI field, it
     activates the Calculate button. */
  g_signal_connect_swapped(
     G_OBJECT(cb_widgets.cf_val), "activate",
     G_CALLBACK(gtk_window_activate_default),
     (gpointer)GTK_WIDGET(window));

  /* The interface is complete, so finally we
     show the top-level window.  This is done
     last or else the user might see the
     interface drawing itself during the short
     time it takes to construct.  It’s nicer
     this way. */
  gtk_widget_show(window);

  /* Enter the GTK Event Loop.  This is where
     all the events are caught and handled.
     It is exited with gtk_main_quit(). */
  gtk_main();

  return 0;
}

/* A utility function for UI construction.  It
constructs part of the widget tree, then
returns its root. */
GtkWidget *create_spin_entry(
          const gchar *label_text,
          const gchar *tooltip_text,
          GtkWidget **spinbutton_pointer,
          GtkAdjustment *adjustment,
          guint digits) {
  GtkWidget *hbox, *eventbox, *spinbutton,
            *label;
  GtkTooltips *tooltip;

  /* A GtkHBox to pack the entry child widgets
     into. */
  hbox = gtk_hbox_new(FALSE, 5);

  /* An eventbox.  This widget is just a
     container for widgets (like labels) that
     don’t have an associated X window, and so
     can’t receive X events.  This is just
     used to we can add tooltips to each
     label. */
  eventbox = gtk_event_box_new();
  gtk_widget_show(eventbox);
  gtk_box_pack_start(GTK_BOX(hbox),
                     eventbox,FALSE, FALSE, 0);
  /* Create a label. */
  label = gtk_label_new(label_text);
```

```
  /* Add the label to the eventbox */
  gtk_container_add(GTK_CONTAINER(eventbox),
                    label);
  gtk_widget_show(label);

  /* Create a GtkSpinButton and associate it
     with the adjustment. It adds/substracts
     0.5 when the spin buttons are used, and
     has digits accuracy. */
  spinbutton = gtk_spin_button_new(
                adjustment, 0.5, digits);
  /* Only numbers can be entered. */
  gtk_spin_button_set_numeric(
           GTK_SPIN_BUTTON(spinbutton), TRUE);
  gtk_box_pack_start(GTK_BOX(hbox),
           spinbutton, TRUE, TRUE, 0);
  gtk_widget_show(spinbutton);

  /* Create a tooltip and add it to the
     EventBox previously created. */
  tooltip = gtk_tooltips_new();
  gtk_tooltips_set_tip(tooltip, eventbox,
                       tooltip_text, NULL);

  *spinbutton_pointer = spinbutton;
  return hbox;
}

/* A utility function for UI construction.  It
   constructs part of the widget tree, then
   returns its root. */
GtkWidget *create_result_label(
          const gchar *label_text,
          const gchar *tooltip_text,
          GtkWidget **result_label_pointer) {
  GtkWidget *hbox, *eventbox, *result_label,
            *result_value;
  GtkTooltips *tooltip;

  /*A GtkHBox to pack entry child widgets into*/
  hbox = gtk_hbox_new(FALSE, 5);

  /* As before, a label in an event box with a
     tooltip. */
  eventbox = gtk_event_box_new();
  gtk_widget_show(eventbox);
  gtk_box_pack_start(GTK_BOX(hbox), eventbox,
                     FALSE, FALSE, 0);
  result_label = gtk_label_new(label_text);
  gtk_container_add(GTK_CONTAINER(eventbox),
                    result_label);
  gtk_widget_show(result_label);

  /* This is a label, used to display the OG
     result. */
  result_value = gtk_label_new (NULL);
  /* Because it's a result, it is set
     "selectable", to allow copy/paste of the
     result, but it's not modifiable. */
  gtk_label_set_selectable(
              GTK_LABEL(result_value), TRUE);
  gtk_box_pack_start(GTK_BOX(hbox),
              result_value, TRUE, TRUE, 0);
  gtk_widget_show(result_value);

  /* Add the tooltip to the event box. */
  tooltip = gtk_tooltips_new();
  gtk_tooltips_set_tip(tooltip, eventbox,
                       tooltip_text, NULL);

  *result_label_pointer = result_value;
  return hbox;
}

/* This is a callback function.  It resets
   the values of the entry widgets, and
   clears the results.  "data" is the
   calculation_widgets structure, which needs
   casting back to its correct type from a
   gpointer (void) type. */
void on_button_clicked_reset(
          GtkWidget *widget,
          gpointer data ) {
  /* Widgets to manipulate. */
  struct calculation_widgets *w;

  w = (struct calculation_widgets *) data;

  gtk_spin_button_set_value(
            GTK_SPIN_BUTTON(w->pg_val), 0.0);
  gtk_spin_button_set_value(
            GTK_SPIN_BUTTON(w->ri_val), 0.0);
  gtk_spin_button_set_value(
            GTK_SPIN_BUTTON(w->cf_val), 0.0);
  gtk_label_set_text(
            GTK_LABEL(w->og_result), "");
  gtk_label_set_text(
            GTK_LABEL(w->abv_result), "");
}

/* This callback does the actual calculation.
   Its arguments are the same as for
   on_button_clicked_reset(). */
void on_button_clicked_calculate(
          GtkWidget *widget,
          gpointer data ) {
  gdouble pg, ri, cf, og, abv;
  gchar *og_string, *abv_string;
  struct calculation_widgets *w;

  w = (struct calculation_widgets *) data;

  /* Get the numerical values from the entry
     widgets. */
  pg = gtk_spin_button_get_value(
                  GTK_SPIN_BUTTON(w->pg_val));
  ri = gtk_spin_button_get_value(
                  GTK_SPIN_BUTTON(w->ri_val));
  cf = gtk_spin_button_get_value(
                  GTK_SPIN_BUTTON(w->cf_val));

  og = (ri*2.597) - (pg*1.644) - 34.4165 + cf;

  /* Do the sums. */
  if (og < 60)
    abv = (og - pg) * 0.130;
  else
    abv = (og - pg) * 0.134;

  /* Display the results.  Note the
     <b></b> GMarkup tags to make it display
     in Bold. */
  og_string =
        g_strdup_printf("<b>%0.2f</b>", og);
  abv_string =
        g_strdup_printf("<b>%0.2f</b>", abv);

  gtk_label_set_markup(
                  GTK_LABEL(w->og_result),
                  og_string);
  gtk_label_set_markup(
                  GTK_LABEL(w->abv_result),
                  abv_string);

  g_free(og_string);
  g_free(abv_string);
}
```

An Introduction to Objective-C
Part 1

D.A. Thomas

This series of articles aims to introduce the Objective-C programming language to readers of C Vu, who are users of C and C++. I will try to show how the language manages to add object-oriented facilities to the low-level features of C in a way that is radically different from that of C++.

Stroustrup, the designer of C++, has been at pains to emphasise that C++ is not just C with an object-oriented extension but rather a new multi-paradigm language that happens to support objects; indeed, recent developments in the language have been more in the direction of generic types ('templates') rather than object-orientation as traditionally conceived. The Standard Template Library, which is now part of the Standard C++ Library, also shows some influence from the functional programming style. In pursuit of this goal, the author and the later standardisers of the language have had no compunction in 'improving' on C by for instance strengthening the type system and changing the namespace rules, with the result that a standards-conforming C source is not guaranteed to perform as the programmer expected after being passed to a C++ compiler.

The designer of Objective-C Brad J. Cox, had other ideas. His goal was to make available some of the facilities of the high-level Smalltalk-80 programming environment to users of C, so as to enhance productivity without taking away the advantages of efficiency, versatility and portability that such a low-level language can offer. The object-oriented part of Objective-C is entirely orthogonal to its C-based component, so that a standard C source is always a perfectly good Objective-C source. Cox's problem, then, was how to extend C in such a way so as to enable powerful and easy object-oriented programming without impinging on the original syntax and semantics of the language. The solution he found is to my mind extraordinarily elegant, though it does involve syntactical conventions that seem quite strange to someone without a Smalltalk background; as a result, it only takes an hour or two's study and practice for a C programmer to learn enough to become proficient at Objective-C.

In his book, *Object-Oriented Programming, an Evolutionary Approach,* Cox addressed what he calls 'the software crisis', namely the tendency of large IT projects to be delivered late, to end up costing much more than their allocated budget, not deliver the intended functionality, and indeed, to fail entirely. Software development is contrasted unfavourably with electronic engineering: software developers keep writing the code over and over again with minor variations to suit the demands of each individual system, while their hardware counterparts are able to create new devices largely by slotting in and connecting components that they have acquired from third parties and are thus much more productive. If development could be largely reduced to putting together systems from *software ICs* with minimal wiring in between, this would go a long way to solving the software crisis, Cox argued. At the time, he knew of only one development system which provided such components at the IC level, and that was Smalltalk, but the use of Smalltalk was limited by the fact that it required its own 'box' or run-time environment to work, and being interpreted rather than compiled, its performance was unspectacular on the computers of the day. Much could be gained if one could program in C and yet make use of libraries that are available to the Smalltalk programmer.

Cox's company, the Stepstone Corporation, brought out an Objective-C compiler in 1983 with a Smalltalk-like library called ICPak(R). ICPak101provided foundation classes, including Object, the root object, and data-structure classes like Sets, Dictionaries, Arrays, Lists and Strings, while ICPak201 had classes to build cross-platform graphical user interfaces; the coroutine library TaskMaster was added later. Richard Stallman and others developed open-source equivalents of the compiler, run-time system and basic class library for the GNU project, starting in 1992. However, arguably the most signigicant event in the history of the language was its adoption in 1988 by NeXT Computer Corporation as the development language of choice for their UNIX-based NexTStep operating system running on their proprietary workstations. They made some extensions to the language, such as Categories and Protocols, and rather then use Stepstone's ICPak libraries, they developed their own, called Foundation and AppKit. NeXT's workstations, though they drew widespread admiration, did not sell well, and so their development environment, now called OpenStep, was ported to other architectures, as NeXT transformed itself into a software company. OpenStep became popular with financial institutions, scientists, and, so I am assured, US intelligence agencies because it could be used to produce powerful applications with a graphical front end in a remarkably short time; one of these applications was the first World-Wide Web browser, written by Sir Tim Berners-Lee at CERN in 1992. Unfortunately, support for multi-platform OpenStep came to an end after NeXT was acquired by Apple in 1996, but the concept still lives in the Cocoa development frameworks on Apple's Mac OS. The open-source GNUstep project provides libraries and tools to do OpenStep-style development on non-Mac OS platforms.

Objective-C's fortunes have been very different from those of C and C++, and, unlike them, it has never become a mainstream development tool. It is beyond the scope of these articles to speculate as to why this might have been so. (I do not believe Dr Cox minds too much, as his idea of exploiting reusable software components has been abundantly vindicated, from Visual Basic custom controls to Python libraries; he has retained his enthusiasm for Smalltalk-style languages and is now a member of the Ruby community.) The closest thing to a standard is Apple's compiler and frameworks, since the vast majority of Objective-C users work either with these or with GNUstep. Subsequent articles will discuss Objective-C as implemented by NeXT/Apple, and the code examples will illustrate the use of the Foundation framework. I have not tested the code on GNUstep but have reason to expect that it will work there with little or no modification.

Availability of Objective-C Resources

Apple's development tools, including their compiler[1], can be downloaded free of charge from their site. As far as I am aware, the original Objective-C compiler and ICPak libraries produced by the Stepstone Corporation are no longer commercially available. The open-source GNU Compiler Collection (GCC) version 3.4 offers a compiler with facilities very similar to Apple's. The Portable Object Compiler (POC) is in reality an Objective-C front-end to various C compilers and is available free from its author, Mr David Stes. Metrowerks CodeWarrior development tools for Mac OS will interoperate with Apple's Interface Builder GUI construction tool and include an Objective-C compiler that will produce code that links with Apple's frameworks. IBM's Visual Age C/C++ Advanced Edition for Mac OS X also has Objective-C support.

D A Thomas

Bibliography and References

Anguish, Scott, Erik Buck and Donald Yacktman, *Cocoa Programming*, SAMS, 2002

Apple Computer Inc., *The Objective-C Programming Language*, available online at `http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/`

Apple Developer Connection `http://developer.apple.com/`

Beam, Michael and James Duncan Davidson, *Cocoa in a Nutshell*, O'Reilly, 2003

Budd, Timothy, *Introduction to Object-Oriented Programming*, 3rd ed., Addison-Wesley, 2001

Cox, Brad J. and Andrew J. Novibilski, *Object-Oriented Programming, an Evolutionary Approach*, 2nd ed., Addison-Wesley, 1991

Cox, Brad, *Planning the Software Revolution*,IEEE Software Magazine, November 1990, available online at `http://www.virtualschool.edu/cox/pub/PSIR`

Cox, Brad, *TaskMaster* available online at `http://www.virtualschool.edu/cox/pub/TaskMaster/`

Davidson, James Duncan and Apple Computer Inc., *Learning Cocoa with Objective-C*, 2nd ed., O'Reilly, 2002

Duncan, Andrew M., *Objective-C Pocket Reference*, O'Reilly, 2002

GNU Compiler Collection `http://gcc.gnu.org`

GNU Objective C Class Library `http://www.cs.rochester.edu/u/mccallum/libobjects` `http://theory.uwinnipeg.ca/gnu/libobjects/libobjects_toc.html`

GNUstep `http://www.gnustep.org/`

Hillegass, Aaron, *Cocoa Programming for Mac OS X*, 2nd ed., Addison-Wesley, 2004

IBM Inc. `http://www-306.ibm.com/software/awdtools/vacpp/features/xlcpp-mac.html`

[references concluded at foot of next page]

1 This compiler is a GCC derivative and also supports a dialect called *Objective-C++*, which allows mixing of C++ and Objective-C source; this is meant to facilitate the porting of legacy applications written in C++ to Cocoa.

C++ Templates - A Simple Example

Rajanikanth Jammalamadaka <rajani@ece.arizona.edu>

This article describes the C++ code for performing basic operations on matrices using templates. That means we can create a matrix of any data type using one line and call the respective functions.

The code listing `opmatrix.h` is the header file in which the matrix class is described. Note that the number of rows and columns is hard coded in the header file. So, in the current code, a matrix of two rows and two columns has been created. These numbers can be changed for matrices of bigger dimensions. Also for convenience's sake, this code works only for square matrices (matrices which have the same number of rows and columns).

The header file defines the matrix as a two dimensional vector. A vector is a container in C++ which is very similar to an array in the C language but is more sophisticated (it manages its own memory and you can call a number of functions to perform useful operations.) The functions `readm()` and `printm()` are used to read in the elements of the matrix and to print the matrix.

Note that the `readm()` and `printm()` functions use the `this` pointer. The this pointer stores the address of the current object. For example, if we declare an object of the matrix class of type `int` like this

```
matrix<int> a;
```

then the `this` pointer will contain the address of the object `a`, i.e. `this = &a`

Therefore, saying `a.readme()` is equivalent to saying

```
for(int i = 0; i < ROWS; ++i)
  for(int j = 0; j < COLS; ++j)
    cin >> (&a)->s[i][j];
```

The overloaded operators `+`, `-`, `*` and `~` are declared as friend functions of the matrix, so that they can access the elements of the matrix (which are private) in order to perform the operations. The overloaded operator `~` is the transpose operator.

The `+` operator operates on two matrices and adds the corresponding elements of the two matrices and prints the result.

The `-` operator is similar to the + operates except that it subtracts the elements of two matrices, instead of adding them.

The `*` operator needs some explanation. Matrix multiplication works only if the number of columns of the first matrix is equal to the number of rows of the second matrix. For example, if we have to multiply matrices `a` and `b`, then the number of columns of matrix `a` must be equal to the number of rows of matrix `b`. In our case this is not a problem because we are dealing with square matrices (whose sizes are fixed once the variables `ROWS` and `COLUMNS` are initialized), so the number of columns of first matrix will always be equal to the number of rows of the second matrix.

In order to understand how the `*` operator works, let us consider a simple example:

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$\text{now, } a*b = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Now, both `a` and `b` are of dimensions 2 by 2.

Therefore, their product will be of dimension 2 by 2. The first element of a*b is obtained by summing the product of the corresponding elements of the first row of matrix a and first column of matrix b:

i.e.

$$(a*b)_{11} = 1*5 + 2*7 = 5 + 14 = 19.$$

$(a*b)_{ij}$ denotes the element at the intersection of i^{th} row and j^{th} column of the product matrix `a*b`. Similarly, $(a*b)_{12}$ is obtained by summing the product of the corresponding elements of the first row of matrix `a` and second column of matrix `b`. Similarly the other two elements can be obtained.

The `~` operator transposes the elements of a matrix. By transpose, we mean interchanging the rows and columns of a matrix. So, a matrix $(A)_{ij}$ after the transpose operation becomes $(A)_{ji}$.

For example the matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

after transposing becomes $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

The above examples may seem to be trivial, but they were purposefully made trivial in order to understand the concepts of basic matrix operations. If the matrices were of large dimensions, it wouldn't be a trivial task to multiply them manually. In the results section, operations are performed on two matrices each of dimension 4 by 4. It is here that operator overloading proves to be most useful. For example, if we have to find the transpose of `a+b*c`, all we need to do is `~(a+b*c)` and store this in a matrix and print the resultant matrix using the `printm()` function.

Also note that each operator accepts a constant reference to the matrix, this is because we want each operator to perform its function without modifying the original matrix which was given to it.

Rajanikanth Jammalamadaka

```
// opmatrix.h - C++ header file

#include <iostream>
#include <vector>

using std::cin;
using std::cout;
using std::vector;
using std::endl;

const int ROWS = 2;
const int COLS = 2;

template<class T>
class matrix {
  // declare a vector of vectors of type T
  vector< vector<T> > s ;
public:
  // Initialize the size of s to ROWS by COLS
  matrix() : s(ROWS, vector<T>(COLS)) {}
  void readm();
  void printm();
  // declare the operators +,-,*,~ as  friends
  // and with return type matrix<T>
  friend matrix<T> operator+<>(const matrix&,
                               const matrix&);
  friend matrix<T> operator-<>(const matrix&,
                               const matrix&);
  friend matrix<T> operator*<>(
          const matrix<T>&, const matrix<T>&);
  friend matrix<T> operator~<>(
                              const matrix<T>&);
};
```

[concluded from previous page]

IBM Inc. `http://www-306.ibm.com/software/awdtools/vacpp/features/xlcpp-mac.html`

Mahoney, Michael and Simson Garfinkel, *Building Cocoa Applications: A Step-by-Step Guide*, O'Reilly, 2003

Metrowerks Inc. `http://www.metrowerks.com/MW/Develop/Desktop/Macintosh/Professional/Mac9.htm`

Objective-C FAQ `http://www.faqs.org/faqs/computer-lang/Objective-C/faq/`

Pinson, Lewis J. and Richard S. Wiener, *Objective-C: Object-Oriented Programming Techniques* , Addison-Wesley, 1991

Portable Object Compiler `http://users.pandora.be/stes/compiler.html`

```
template<class T>
void matrix<T>::readm() {
  for(int i = 0; i < ROWS; i++)
    for(int j = 0; j < COLS; j++)
      cin >> this->s[i][j];
}

template<class T>
void matrix<T>::printm() {
  for(int i = 0; i < ROWS; i++) {
    for(int j = 0; j < COLS; j++)
      cout << this->s[i][j] << "\t";
    cout << endl;
  }
}

template<class T>
matrix<T> operator+(const matrix<T>& a,
                    const matrix<T>& b) {
  // declare a matrix temp of type T to store
  // the result and return this matrix
  matrix<T> temp;
  for(int i = 0; i < ROWS; i++)
    for(int j = 0; j < COLS; j++)
      temp.s[i][j] = a.s[i][j] + b.s[i][j];
  return temp;
}

template<class T>
matrix<T> operator-(const matrix<T>& a,
                    const matrix<T>& b) {
  matrix<T> temp;
  for(int i = 0; i < ROWS; i++)
    for(int j = 0; j < COLS; j++)
      temp.s[i][j] = a.s[i][j] - b.s[i][j];
  return temp;
}

template<class T>
matrix<T> operator*(const matrix<T>& a,
                    const matrix<T>& b) {
  matrix<T> temp;
  for(int i = 0; i < ROWS; i++) {
    for(int j = 0; j < COLS; j++) {
      temp.s[i][j] = 0;
      for(int k = 0; k < COLS; k++)
        temp.s[i][j] += a.s[i][k] * b.s[k][j];
    }
  }
  return temp;
}

template<class T>
matrix<T> operator~(const matrix<T>& trans) {
  matrix<T> temp;
  for(int i = 0; i < ROWS; i++)
    for(int j = 0; j < COLS; j++)
      temp.s[j][i] = trans.s[i][j];
  return temp;
}

// matrix.cpp - C++ Source file
#include "opmatrix.hpp"

int main() {
  matrix<int> a,b,c;  // we can also declare
   // matrices of type int, float, double etc.
  cout << "Enter matrix a:" << endl;
  a.readm();
  cout << "a is:" << endl;
  a.printm();
  cout << "Enter matrix b:" << endl;
  b.readm();
  cout << "b is:" << endl;
  b.printm();

  c = a + b;
  cout << endl << "Result of a+b:" << endl;
  c.printm();
  c = a - b;
  cout << endl << "Result of a-b:" << endl;
  c.printm();
  c = a * b;
  cout << endl << "Result of a*b:" << endl;
  c.printm();
  cout << '\n' << "Result of a+b*c is:" << '\n';
  (a+b*c).printm();
  c = ~(a+b*c);
  cout << '\n' << "Result of transpose of "
       << "a+b*c is:" << '\n';
  c.printm();
  return 0;
}

// Output

> g++ matrix.cpp -o matrix
> matrix

Enter matrix a:
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
a is:
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16

Enter matrix b:
17 18 19 20
21 22 23 24
25 26 27 28
29 30 31 32
b is:
17      18      19      20
21      22      23      24
25      26      27      28
29      30      31      32

Result of a+b:
18      20      22      24
26      28      30      32
34      36      38      40
42      44      46      48

Result of a-b:
-16     -16     -16     -16
-16     -16     -16     -16
-16     -16     -16     -16
-16     -16     -16     -16

Result of a*b:
250     260     270     280
618     644     670     696
986     1028    1070    1112
1354    1412    1470    1528

Result of a+b*c is:
61189   63786   66383   68980
74025   77166   80307   83448
86861   90546   94231   97916
99697   103926  108155  112384

Result of transpose of a+b*c is:
61189   74025   86861   99697
63786   77166   90546   103926
66383   80307   94231   108155
68980   83448   97916   112384
```

XML as a Model-View-Controller System for Documents

Matthew Strawbridge

Models and Views

The Model-View-Controller (MVC) paradigm is well known by programmers as a way of separating the logical internals of a software system (the **model**) from the code concerned with presenting information to the user (the **view**). Any framework that co-ordinates the interaction between models and views is termed a **controller**. This scheme has been adopted by most modern development frameworks, since it helps software to grow over time in a flexible way, and helps to encapsulate changes. You can add new views to existing models without having to change the models themselves, and business logic can be modified without you needing to change the way it is presented to the user.

Many programmers would rightly condemn code that comprised a mish-mash of logic and presentation, but they are content to produce and consume documents that do precisely this. Memos, technical notes, meeting minutes: these are the bread and butter of the professional Software Engineer, and yet most documents are simply dumped into a word processor and left to stagnate. In this article I will describe how an MVC approach to the generation of documents can yield the same benefits that are traditionally seen with this approach to software design, and will introduce some XML [1][2] tools that can support this method. Finally, I will look at some of the alternatives to XML that could achieve the same separation of concerns.

The Problem

You may think that MVC is overkill for documents – after all, a memo is simply text; there is only one view, and that's the document you're looking at. However, what if you want to put a copy on your company intranet? I daresay your word processor has a 'save as HTML' facility. Good. What if you want to make all document references into hyperlinks; or to change the copyright text in a number of documents you've already saved as HTML; or to radically change the style of every memo. All less good.

As an example, let's take a simple document type with which we're all familiar: an ACCU book review. We already know about two views that exist on these documents: the magazine text (lets assume it's Rich Text Format), and the online review on the ACCU website (in HTML). Remember that, as well as having two different formats, the reviews can also have different content, since some reviews have a short version published in C Vu and an extended version on the Web site.

XML Solution

Model

The starting point for our XML solution is to develop a Document Type Definition that describes the format of the raw information from which we will generate our documents. Strictly speaking, we could bypass this step, but then we would have no way of validating the input document – we would just try to process whatever was given. Note also that XML Schema [3][4] could have been used to provide a more detailed and robust way of validating input documents.

```
<!ELEMENT bookreview (bookdetails, reviewdetails,
                      reviewbody)>

<!ELEMENT bookdetails EMPTY>

<!ATTLIST bookdetails title          CDATA #REQUIRED
                      author         CDATA #REQUIRED
                      isbn           CDATA #REQUIRED
                      publisher      CDATA #REQUIRED
                      pages          CDATA #REQUIRED
                      priceinpounds  CDATA #REQUIRED
                      priceindollars CDATA #REQUIRED>

<!ELEMENT reviewdetails EMPTY>

<!ATTLIST reviewdetails date     CDATA #REQUIRED
                        reviewer CDATA #REQUIRED>

<!ELEMENT reviewbody (para+)>

<!ELEMENT para (#PCDATA)>

<!ATTLIST para filter (shortonly | longonly)
#IMPLIED>
```

This simple DTD just says which XML elements are allowed in a book review, and which attributes each of them may contain. Here is an example review adhering to this DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BookReview SYSTEM "BookReview.dtd">
<bookreview>
  <bookdetails title="How to Write a Book Review"
               author="B. Worm"
               isbn="0-123-45678-9"
               publisher="A. B. Cee Ltd."
               pages="123"
               priceinpounds="12.50"
               priceindollars="16.00"/>
  <reviewdetails date="2003-11-19"
                 reviewer="Matthew Strawbridge"/>
  <reviewbody>
    <para>This is an excellent book that tells you
      all about how to review books.</para>
    <para filter="longonly">You should buy this
      book because..., and finally because it's
      two inches thick so it must be good.</para>
    <para filter="shortonly">Buy this book.</para>
  </reviewbody>
</bookreview>
```

This example is a hypothetical review of the book *How to Write a Book Review* by B. Worm, which was supposedly reviewed by me on the 19th November 2003. Note that the final two paragraphs provide a long description for the Web and a short description for the magazine respectively.

Views

From this single source, we want to generate the following two documents. Don't worry if you're not familiar with RTF or XHTML – the precise content that gets generated in each case is not that important; the point is that radically different target documents need to be generated from a single source.

RTF for Print

```
{\rtf

{\b How to Write a Book Review}
\par By B. Worm
\par A. B. Cee Ltd. ISBN: 0-123-45678-9, 123pp, UKP
12.50 [$16.00 (1.28)]
\par

\par Reviewed by Matthew Strawbridge on 2003-11-19
\par This is an excellent book that tells you all
about how to review books.
\par Buy this book.}
```

XHTML for Web

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1/DTD/
                                  transitional.dtd">
<html><head><title>
  Book Review -
  How to Write a Book Review</title></head><body>
```

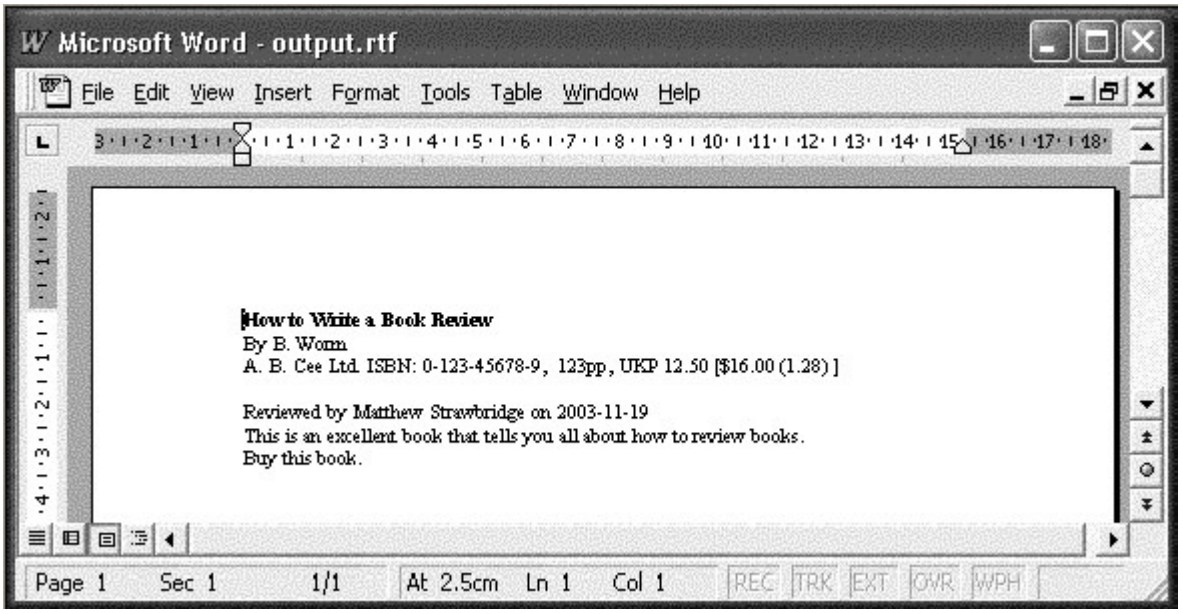


Figure 1: RTF output

```
  <p><em>How to Write a Book Review</em></p>
  <p>By B. Worm</p>
  <p>A. B. Cee Ltd.
     ISBN: 0-123-45678-9,
     123pp,
     UKP 12.50
     [$16.00 (1.28)]
  </p><hr/>
  <p>Reviewed by Matthew Strawbridge on
     2003-11-19</p><hr/>
  <p>This is an excellent book that tells you all
     about how to review books.</p>
  <p>You should buy this book because..., and
     finally because it's two inches thick so it
     must be good.</p>
</body></html>
```

Controllers

The main benefit of using XML to capture the model is the ease with which it can be parsed, and reshaped into different formats. This is done using XSLT [5][6], the Extensible Stylesheet Language for Transformations. An XSLT stylesheet is an XML document that uses pattern matching rules to transform an XML base document into some other form. Here are the XSLT stylesheets required for transforming our Bookreview base document into the two output types.

XSLT for Converting Bookreview to RTF

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="text"/>

  <!– RTF should not have any unintentional blanks,
      so strip them –>
  <xsl:strip-space elements="*"/>

  <!– Template that matches the outer bookreview
      element and constructs an RTF document from
      it –>
  <xsl:template match="bookreview">
        {\rtf<xsl:apply-templates/>}</xsl:template>

   <xsl:template match="bookdetails">
<!– Make the title bold –>
{\b <xsl:value-of select="@title"/>}
\par By <xsl:value-of select="@author"/>
\par <xsl:value-of select="@publisher"/>
      ISBN: <xsl:value-of select="@isbn"/>,
      <xsl:value-of select="@pages"/>pp,
      UKP <xsl:value-of select="@priceinpounds"/>
      [$<xsl:value-of select="@priceindollars"/>
      (<xsl:value-of select="@priceindollars
                             div @priceinpounds"/>) ]
\par
  </xsl:template>

  <xsl:template match="reviewdetails">
\par Reviewed by <xsl:value-of select="@reviewer"/>
     on <xsl:value-of select="@date"/>
  </xsl:template>

  <xsl:template match="para">
    <!– Include paragraphs only if they either
        have no filter, or if the filter is set
        to 'shortonly' –>
    <xsl:if test="not(@filter) or
          @filter='shortonly'">
\par <xsl:value-of select="."/>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

XSLT for Converting Bookreview to XHTML

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="xml"
    doctype-public="-//W3C//DTD XHTML 1.0
                                Transitional//EN"
    doctype-system="http://www.w3.org/TR/
             xhtml1/DTD/xhtml1/DTD/transitional.dtd"
    encoding="ISO-8859-1"
    indent="no"/>

  <!– Template that matches the outer bookreview
      element and constructs an XHTML page from it–>
  <xsl:template match="bookreview">
    <html>
      <head>
        <title>
         Book Review -
         <xsl:value-of select="bookdetails/@title"/>
        </title>
      </head>
```

```
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="bookdetails">
    <p><em><xsl:value-of select="@title"/></em></p>
    <p>By <xsl:value-of select="@author"/></p>
    <p>
       <xsl:value-of select="@publisher"/>
       ISBN: <xsl:value-of select="@isbn"/>,
       <xsl:value-of select="@pages"/>pp,
       UKP <xsl:value-of select="@priceinpounds"/>
       [$<xsl:value-of select="@priceindollars"/>
       (<xsl:value-of select="@priceindollars
                    div @priceinpounds"/>)
       ]
    </p>
    <hr/>
  </xsl:template>

  <xsl:template match="reviewdetails">
    <p>Reviewed by <xsl:value-of select="@reviewer"/>
       on <xsl:value-of select="@date"/></p>
    <hr/>
  </xsl:template>

  <xsl:template match="para">
    <!— Include paragraphs only if they either have
        no filter, or if the filter is set to
        'longonly' —>
    <xsl:if test="not(@filter) or
            @filter='longonly'">
      <p>
        <xsl:value-of select="."/>
      </p>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Performing the Transformation

To actually apply these rules to the base document, an XSLT processor must be used. There are a number of these available as open source projects on the Web, the two most well-known being Xalan [7] and Saxon [8].

These templates really specify only the minimum amount of information that is needed to generate the final documents – there is very little wasted effort. Imagine deciding instead to solve the program 'programatically', and parse and transform the XML in a C++ or Java program. Indeed, XSLT is a lot more powerful than these simple examples show.

Further Improvement

These files are only meant as a demonstration of the method, and there are many improvements that could be made to create a better system for real-world use:

- `priceindollars` should be `#IMPLIED` (meaning optional), rather than `#REQUIRED`. In fact, the system should be changed to cope with various types of currency.
- It would be useful to add an enumerated summary, with options such as 'highly recommended', 'recommended', 'not recommended' and 'recommended with reservations'.
- Typically, you would want to batch process a number of reviews at once (or even every existing review, in the case of applying an updated template to the Web site). There should probably be an outer wrapper, such as `<reviewset>`, which can contain one or more `<review>` elements, and the XSLTs should handle generating either one long document, or a separate document for each review, from this collection of reviews. An alternative would be to set up a make file, perhaps using Ant [9], which includes support for XSLT transformations.
- The dates need translating from `YYYY-MM-DD` to a format that is more pleasant to read.
- In many cases it may be better to write a single XSLT that will convert from your bespoke format into Docbook, for which there are already some comprehensive stylesheets for conversion into many formats including HTML and PDF.

As it is customary to say is such situations, these improvements are left as an exercise for the reader.

Alternatives

Word Processor Styles

Most modern word processors support *styles*, whereby a set of properties can be assigned to segments of text. These styles can then be updated, and the updates will be automatically applied to all text having that style. While this follows the 'separation of concerns' regarding content and presentation, there are several key areas in which the XML method is to be preferred. The main difference is that styles do not transform the contents of the document, so our example using `longonly` and `shortonly` attributes for paragraphs could not be implemented without the use of macros. It would also be difficult to regenerate a batch of documents if the template changes, especially if multiple *Save As* formats were needed.

Microsoft Word 2003

I have read about the XML support in Microsoft Office 2003, but haven't used it myself. I would be interested to know if anyone uses it in a similar fashion to that described here – at the very least it promises to be a more user-friendly way of populating the raw XML files than simply using a text editor.

[concluded at foot of next page]

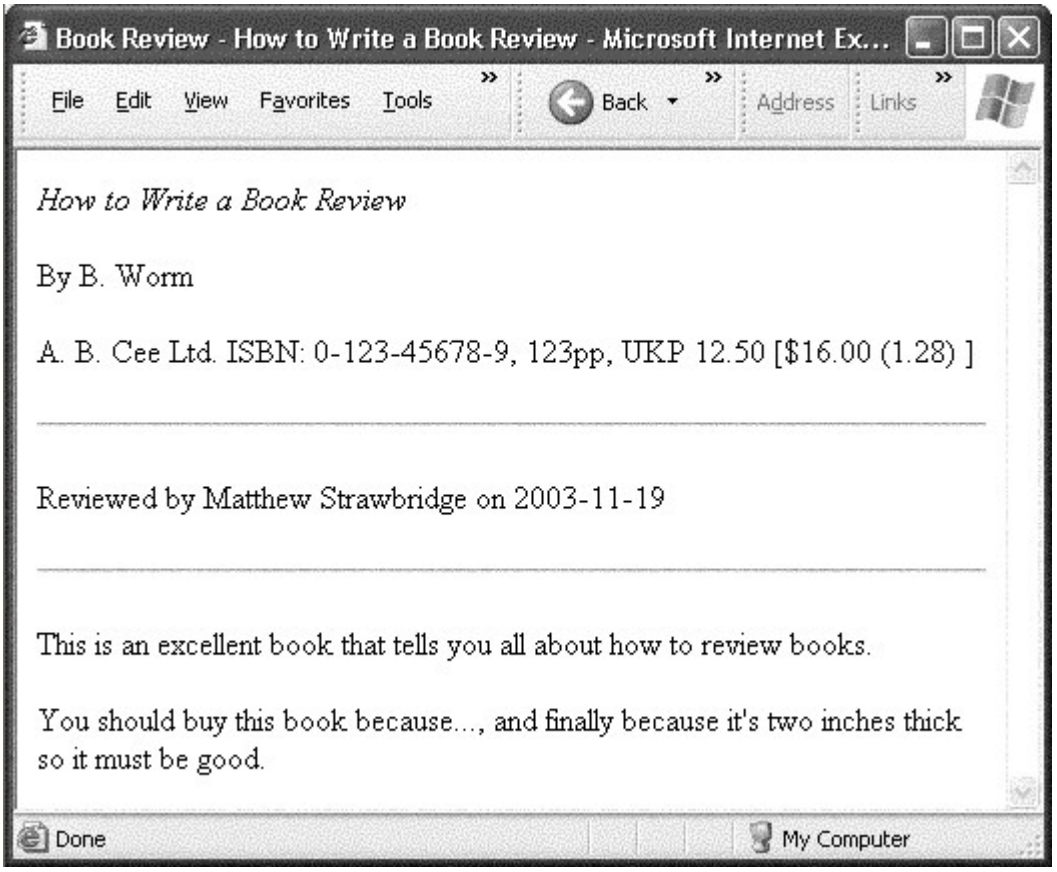


Figure 2: HTML output

Introduction to C# - Part 2

Mike Bergin <mijobee@xinjen.com>

Welcome Back

Welcome to the second in a series of articles introducing the C# programming language. In the previous issue the basics of the language such as variables, methods and classes were covered. In this issue classes are covered in more detail highlighting an important feature called *inheritance*. If you are just tuning in, and have some programming experience, this article should still be digestible.

Classes, Objects and Types

A Touch of Class

One of the most common descriptions of classes that I have found is, "*A class is a template for creating objects frequently used to model real world entities such as a bank account or automobile*." This is an excellent and concise description of a class but for me it helped to relate the idea of an "object template" to something I was already familiar with. Another place templates are found is in word processing applications and as it turns out this is a good analogy.

Document templates specify fonts for different sections, nice borders and graphics as well as the overall layout. A resume template might define sections for contact information, objective, work history and education, including appropriate formatting for headings and body text. When I need to whip up a new resume I open my word processing application and create a new document from the resume template. Now all that's left to do is fill in the information that is unique to me and I have a shiny new resume.

A class can also be created that specifies the different sections, borders, graphics and layout for a resume.

```
public class Resume {
  public string Name, PhoneNumber, Objective;
  ...
  public void Print() {
    ...
  }
  ...
}
```

This `Resume` class contains `string` member variables that store text for the same sections that the document template provides. It also has a `Print` method that might print the resume to the screen for review or to a printer.

Now that we have our template we follow a process similar to creating our resume with the word processor. First we need to create a new object from the class just as we created a new document from the word processor's resume template.

```
Resume myResume = new Resume();
```

Now that we have our handy new object we need to fill in our information.

```
myResume.Name = "Mike Bergin";
myResume.PhoneNumber = "555-2234";
myResume.Objective = "Make tons of money!";
```

The resume now contains all of my relevant personal information so now it's time to print.

```
myResume.Print();
```

There are a few important points to note about this example. The first point is that when we modify an object, or document, it does not modify the template. The second point is if we create two objects, or two documents, from the same template, changes to one of them are not reflected in the other. For example if I enter my name into one object, or document, it does not appear in yours, the two are unique entities.

What Type of Person Are You?!

Classes are also called *types*. This use of the word type is similar to how people use it to categorize each other. For example a person confronting someone they are upset with might say, "I didn't know you were that type of person." They are referring to the actions, feelings or thoughts of someone, just as calling a member variable a type is a categorization of the methods, actions, and data of that piece of software. Assuming that the argument is between a wife and her cheating husband the following example provides a simple representation of "that type of person" in software.

```
class Cheater {
  string Name, SpouseName, MistressName;
  void PerformUnfaithfulAct() {
    // Outside the scope of this article
  }
}
```

This snippet of code defines a new class, or type, named `Cheater` containing three `string` member variables and one method. Instances of this class could be created to represent all the cheating husbands of the world, all created from the same template, but each unique.

```
void CreateCheatingHusbands() {
  Cheater bob = new Cheater();
  bob.Name = "Bob Smith";
  Cheater john = new Cheater();
  john.Name = "John Doe";
}
```

Inheritance

Consider a scenario where you need to write a textbook describing each star in the solar system for an advanced astronomy course. The descriptions must be extremely precise, covering information that might even change during the lifetime of the book due to new discoveries in modern science. The first thing you would most likely do is identify the characteristics

[continued on next page]

Final Word

Many software developers have a real loathing for any form of documentation. ACCU members, generally being a well-read bunch, may have less of an aversion, but one thing is clear – if you dislike writing documents, then you'll really hate having to make minor updates to several hundred of them by hand. By separating out the content (model) from the presentation (view), and creating reusable templates to generate one from the other, maintenance of an archive of documents is greatly simplified. XML and XSLT can be used to implement such an MVC treatment of documents.

As with most things in the world of computers, there is more than one way to string a cat (or should that be to `cat.toString()`?) I am not advocating that all documents should be written in this way, but for cases where you have lots of similar documents that do, or may, need to be rendered in more than one format, this technique should save a lot of time in the long-run at the expense of a little work up-front. Now, where have I heard that before?

Matthew Strawbridge

References

[1] *XML*. `http://www.w3.org/XML/`
[2] Elliotte Rusty Harold, W. Scott Means. *XML in a Nutshell*, O'Reilly, 2002
[3] *XML Schema*. `http://www.w3.org/XML/Schema/`
[4] Eric van der Vlist. *XML Schema*, O'Reilly, 2002
[5] *XSLT*. `http://www.w3.org/Style/XSL/`
[6] Doug Tidwell. *XSLT*, O'Reilly, 2001
[7] *Xalan*. `http://xml.apache.org/xalan/`
[8] *Saxon*. `http://saxon.sourceforge.net`
[9] *Ant*. `http://jakarta.apache.org/ant/index.html`

shared by all stars. Next you might categorize the different types of stars based on some common characteristics, for example by size and temperature grade. Now that the generic characteristics and groups of similar stars have been identified you might begin writing.

The first chapters of your book would most likely cover the common characteristics of all stars, which you identified in the initial phases of analysis. These chapters would contain very general information such as the chemical elements that stars are composed of, however it would not contain the amounts because they are different for each star. Next you would pick a category of star, such as white dwarfs, describing only what makes that group of stars unique. You would be careful not to provide any redundant information simply by referring back to previous sections. Once you had covered each category you would cover each specific star, providing information about what makes it unique among its group such as its name, coordinates and precise chemical composition.

As new discoveries are made some of the information in the textbook may become invalid. For example an astronomer may discover a new element present in all stars. Updating your textbook to reflect this information would be relatively simple because the topic of composition was only covered once and then referred back to, so the text only needs to be updated in one place. Another side effect of this approach is that it is much quicker to describe each star in relation to a set of common characteristics instead of describing each in isolation.

Source code is essentially a description of the data and behavioural composition of an application. When a program is written using a language that allows software to be expressed by modelling real world entities, such as C#, developers follow the same basic process we did in writing the textbook. In order to illustrate this let's apply the same steps to the development of an address book application used to manage contact information.

The first step is to identify the common characteristics of all contacts managed by the application. We perform our analysis by examining a few examples of contacts the application will need to manage. One example is our Aunt Marilyn, who we find has a name, phone number address, birthday and email address. Another example we examine is our favourite pizza restaurant, and we find that it has a name, phone number, address and website. Comparing these two examples we find that they both have a name, phone number and address. Now that we have identified the characteristics common to all contacts we can write this description in C#.

```
public class Contact {
  public string Name, PhoneNumber, Address;
}
```

The next step that we took in writing our textbook was to identify groups of similar items so we do that next. By performing our analysis of the characteristics shared by all contacts we have actually identified these groups, *people* and *businesses*. Before writing our description of these two groups we must keep in mind that we don't want to provide any redundant information, we only provide what makes them unique.

```
public class Person : Contact {
  public string EmailAddress, BirthDay;
}

public class Business : Contact {
  public string WebSite;
}
```

You may notice that the name of the `Contact` class follows the name of each new class separated by a colon. In C# this notation is used to indicate that the reader should refer to another section of the source code for more information. In this case we are saying that the description of `Person` is the sum of the description provided and the description of `Contact`. In our textbook we would have appended a superscript numeral to a word and then indicated at the bottom of the page that the reader is to refer back to a particular section to provide this same indication.

The sun is composed[1] of 91.2% Hydrogen, 8.7% Helium

…

1. Refer back to Chapter 12 Composition for more information.

Now that we have identified the common characteristics of all contacts and the groups, let's review exactly what we have. First we found that all contacts have a name, phone number and address. Once we had identified these traits we described them in the `Contact` class. The next product of our analysis was the identification of two groups of contacts, namely people and businesses. We found that all people have an email address and birthday in addition to what was already described in the `Contact` class, and described this in the `Person` class. Businesses were found to all have a website in addition to what was already described in the `Contact` class, so we described this in the `Business` class. Whew! Now that we have the hard part done its time to describe each unique entry in our address book.

In software written in C#, classes are used to describe the groups of items we are working with and objects are used to represent the members of those groups. So in this particular case we now create instances of the `Person` class for each person in our address book and instances of the `Business` class for each business.

```
Person auntMarilyn = new Person();
auntMarilyn.Name = "Marilyn Bergin";
auntMarilyn.EmailAddress =
                  "marilyn@somewhere.com";
...
Business pizzaPlaza = new Business();
pizzaPlaza.Name = "Pizza Plaza";
pizzaPlaza.WebSite = "www.pizzaplaza.com";
...
```

The execution of the code in the last snippet deserves a bit of explanation so let's run through it line by line to see what's happening. On the first line a new instance of the `Person` class is created and stored in a variable named `auntMarilyn`. The next line stores the string `Marilyn Bergin` in the `Name` member variable of the `auntMarilyn` object. When the computer reads this line of code it looks at the `Person` class for a description of a member variable named `Name`. The computer doesn't find it there but the class indicates that the computer should also look at the `Contact` class for more information, where it does find a member variable named `Name`. Next the string `marilyn@somewhere.com` is stored in a member variable named `EmailAddress`. Again the computer looks back to the `Person` class for a description of a member variable and this time it finds it. The same process is followed for the lines that deal with the instance of the `Business` class. The computer looks in the `Business` class for information and if it doesn't find it refers to the `Contact` class as indicated.

Inheritance is the ability to describe a section of code in relation to another section of code. As we saw in the `Person` and `Business` classes, C# provides the inheritance notation, the colon followed by the name of the generic class, to indicate that everything from that class should be pulled into a new class that also adds a few items of its own. So when working with an instance of the `Person` class we know it not only contains the member variables defined in its own class definition but also those defined in the `Contact` class's definition as well. This is referred to as one class inheriting from another class. In this scenario the `Contact` class is referred to as the *base class* or *superclass* and the `Person` class is called the *subclass* or *derived class*.

Inheritance is an extremely powerful feature that provides a number of benefits when used properly. The two main aspects of inheritance to consider are that it cuts down on the sheer amount of source code and that many classes can inherit from the same class. Cutting down on the amount of source code helps us to realize the one of the big promises of OOP, code reuse. When we reuse code we cut down on development time because we don't need to write as much, and maintenance is also cut down, which has been shown to be more expensive than development in many cases. The second point to note is that many classes can share the same base class, and as we will see in the following sections this is another extremely powerful aspect of inheritance.

There is a fair amount of terminology used when discussing inheritance and software that supports this feature so before we continue it may be helpful to provide some definitions. As we have seen a class that is inherited from is referred to as a *base class* or a *superclass*. A class that inherits from another is called a *derived class* or a *subclass*. Also in both of these scenarios the word type might be used interchangeably with the

word class such as *derived type* instead of *derived class*. When a class inherits from another class the act of doing so is referred to as *subclassing*. Inheriting from another and providing additional members or other customizations is called *specialization*. You don't need to commit all of these to memory, just know they are here if you see some confusing jargon later.

Leveraging Inheritance

Now that we have explored the basic concept of inheritance and how it is supported by C# we can move on to some other features that are implemented through the use of inheritance.

Polymorphism

Polymorphism is the ability of a variab le to reference an instance of the variable's delcared type *or a subclass of it*. For example, a variable of type `Contact` could be used to store an instance of either the `Person` or `Business` classes.

```
Contact contact = new Person();
```

When an instance of a subclass is referenced using a variable of its base class's type, only the members declared in the base type are accessible. In the preceding example if we had tried to reference the `EmailAddress` member variable we would have received an error when compiling the code. The reasons for this are purely mechanical; only instances of `Contact` or one its subclasses may be stored in the variable because we are only guaranteed that the members defined in the `Contact` class are supported.

This makes sense because `Person` inherits all members defined in the `Contact` class so it exposes the same methods, properties, variables, etc. One catch to this is that only the members defined by the variable's type may be used no matter what type of object is actually stored in the variable. Simply put, if we store an instance of the `Person` class in a `Contact` variable we can only access members declared in the `Contact` class. You might be thinking that the compiler knows we are storing a `Person` object in the variable because the code is on the same line. This is a valid point in this situation but consider if we used polymorphism when calling a method.

```
public void Close(Contact contact) { ... }
public void Main() {
  Close(new Person());
}
```

In the body of the `Close` method we couldn't know what type had actually been passed in, just that it supports all of the members defined in the `Contact` class. This is compounded by the fact that code might be introduced at a later time that calls this method, passing an instance of a class that didn't exist when the code containing the `Close` method was written.

Specialization

Inheritance allows us to pull one class's implementation into another, which as we've seen can be extremely useful. There are many times however when the inherited implementation doesn't quite meet our needs, luckily C# allows us to effectively replace inherited members with our own implementations. Not all members may be replaced, C# requires class authors to declare that a member may be replaced using the `virtual` keyword. Subclasses may replace `virtual` members by providing one with the same signature but replacing `virtual` with the `override` keyword.

```
public class Contact {
  public virtual void Save(Connection conn) {
    // Save member variable values
  }
}

public class Person : Contact {
  public override void Save(Connection conn) {
    base.Save();
    // Save extended member variable values
  }
}
```

In this code snippet we added a new `virtual` method named `Save` to the `Contact` class that saves its member variables. Classes that inherit this method must replace it with their own, which saves any new members they introduce in addition to the inherited members. In order to simplify this task, C# allows subclasses to explicitly refer to members of their superclass using the `base` keyword, even when it's been overridden. The `base` keyword allows subclasses to not just replace a member but to reuse it, in effect customizing the existing member for its specific purpose.

Combining Polymorphism and Specialization

Polymorphism and specialization can be combined in powerful ways to develop extremely dynamic software. Here we will make use of the new `Save` method introduced in the previous example in an attempt to showcase some of the power available to C# programmers.

```
public void Persist(Contact contact) {
  Connection conn = GetConnection();
  Contact.Save(conn);
  CloseConnection(conn);
}

public static void Main() {
  Person auntMarilyn = new Person();
  auntMarilyn.EmailAddress =
                    "marilyn@somewhere.com";
  ...
  Business pizzaPlaza = new Business();
  pizzaPlaza.WebSite = "www.pizzaplaza.com";
  ...
  Persist(auntMarilyn);
  Persist(pizzaPlaza);
}
```

In the example above we defined a method named `Persist` that saves any instance of the `Contact` class or one of its subclasses. When the instance of the `Person` class is passed to the `Persist` method the `Save` method implemented in the `Person` class is called, causing all of its variables to be saved. When the instance of the `Business` class is passed however, only the variables declared in the `Contact` class are saved because the `Business` class did not customize the `Save` method to include the variables it introduced. As we can see the `Persist` method doesn't care what the actual type stored in the variable is, only that it supports the `Save` method.

Interfaces

The term *interface* refers to the mechanisms provided to interact with a class, such as the member variables, methods and properties it exposes.

An interface is declared in much the same way as a class is declared, except that it does not provide any implementation, the parts between the opening `{` and closing `}`. A class can declare that it implements a particular interface and then provide implementations of all the interface's members. Variables may declare that they hold instances of a particular interface just as a variable may declare that it contains an instance of a particular class. Interfaces cannot be instantiated, but classes that implement the interface may be stored in one of these variables.

In the section on polymorphism the `Contact` class defined a method named `Save` that saved the values stored in the object's variables to a database. Instead of this method being declared in the `Contact` class it could have been declared in an interface instead. This might be the case if you were using a third party persistence framework for example. The interface provided by the third party framework would be implemented by objects that need to be persisted, and the framework would do the rest. In this scenario the `Contact` class would most likely implement the interface and provide default implementations for each interface member. Let's assume the interface from the persistence framework is defined as follows.

```
public interface IPersistable {
  public void Save(Connection conn);
  public void Read(Connection conn);
}
```

Assuming this is the interface required by the persistence framework, we would need to make a few changes to our classes if they are to take advantage of this new persistence framework. First we would change the `Contact` class to implement the interface.

```
public class Contact : IPersistable {

  ...

  public void Save(Connection conn) {
    // Save to the database...
  }

  public void Read(Connection conn) {
    // Read from the database...
  }
}
```

There are a few points to note about this example. First, the notation used to declare that a class implements an interface is exactly the same as for declaring inheritance. If a class both inherits from a class and implements an interface then the interface name follows the base class name separated by a comma. For example if the `Contact` class inherited from a class named `Widget` then it would be modified as shown below.

```
public class Contact : Widget, IPersistable {
  ...
}
```

It is legal for a single class to implement multiple interfaces, in which case each additional interface would be listed in the same comma delimited fashion. Interface members implemented by a class are implicitly virtual. Now that we have implemented this interface in our `Contact` class, let's look at the changes we need to make to the `Person` class.

```
public class Person : Contact {
  public override void Save(Connection conn) {
    base.Save(conn);
    // Save members defined in Person
  }

  public override void Read(Connection conn) {
    base.Read(conn);
    // Read members defined in Person;
  }
}
```

Notice that we did not explicitly declare that the `Person` class implements the `IPersistable` interface. The reason for this is that interface implementation is transitive, meaning that because the `Person` class inherits from the `Contact` class it implicitly implements the interface, so declaring it again would be redundant. In the methods that override the inherited implementations of the interface members we again make use of the base class's implementation, being careful to not reproduce any code we can reuse.

Now that we have examined how an interface is implemented, let's take a look at how these changes might be used in the persistence framework. The following code snippet might exist in a class that facilitates the persistence of objects using the framework.

```
public class PersistenceManager {
  public void Persist(IPersistable subject) {
    Connection conn = GetConnection();
    Subject.Save(conn);
    ReleaseConnection(conn);
  }
}
```

This illustrates that the important point that interfaces, like inheritance, allow for polymorphism. The `PersistenceManager` class doesn't care how the `Save` method is implemented by the instance passed into the method, or even what type it actually is. All that matters is that the parameter implements the proper interface.

Abstract Classes

Abstract classes are a mix between using inheritance to reuse code and leveraging polymorphism with interfaces. An abstract class may contain both concrete members and interface style members that don't declare any implementation. The concrete members are inherited by derived classes just as normal methods are and can use the `virtual`/`override` keywords in the same way as well. The interface style members require that subclasses provide an implementation for them, just as implementing an interface require the implementers to do. Essentially that's all there is to it, so let's go over a quick example.

The example we reviewed in the section on interfaces could have also been implemented using abstract classes. Implementing the persistence framework using inheritance allows more power to be embedded into the `Contact` class itself. In many cases frameworks that use abstract classes instead of interfaces take this approach and sometimes are able to minimize the amount of work on the part of the user of the framework. In this new example we will be inheriting from an abstract class defined as shown below.

```
public abstract class DatabaseObject {
  public void Save(Connection conn) { ... }
  public void Read(Connection conn) { ... }
  public abstract bool IsDirty();
  public abstract void ClearDirty();
}
```

The declaration of this class includes a keyword that we haven't encountered before, `abstract`. The `abstract` keyword must appear in the declaration of a class that has any abstract members. This same keyword appears in the two interface style member declarations `IsDirty` and `ClearDirty`. Just like with interfaces, no implementation is provided, requiring subclasses to provide it. The `abstract` keyword must be used in these declarations because this is not an interface, so it must be clearly stated if a member is abstract.

In this new framework the `Save` and `Read` methods are actually implemented for us, all that we as the users of the framework need to do is implement a method that indicates if the object is dirty, meaning that data has been modified since it was last saved to the database. The framework might periodically check to see if the `IsDirty` method returns true and if so call the `Save` method, followed by a call to the `ClearDirty` method to indicate that the values have been saved. Tracking when changes were made to the object is left up to the object itself, however they are now automatically saved to the database by simply returning `true` from a method call. In closing, the following example shows how the `Contact` class might look if it used this inheritance based framework.

```
public class Contact : DatabaseObject {

  public bool hasChanged;

  public override bool IsDirty() {
    return hasChanged;
  }

  public override void ClearDirty() {
    hasChanged = false;
  }
}
```

See You Next Time

In this issue we discussed some of the fundamental features of classes. In the next issue we will cover a few more class-related features, as well as the type system supported by C#. If you read the first article and are wondering what happened to the Address Book application, we will eventually get to incorporate everything we've covered. See you next time!

Mike Bergin

Reviews

Bookcase

Collated by Christopher Hill
<accubooks@progsol.co.uk>

A Note from Francis

The book reviews this time seem to exhibit one of those odd statistical quirks; there seems to be a completely disproportionate number of reviews that can best be classified under the headings 'Methodologies'. Perhaps it is a sign that we are moving away from concerns with learning to program towards being concerned with how those skills can be appropriately applied in the software development process. What do you think?

Francis

The following bookshops actively support ACCU (the first three offer a post free service to UK members – if you ever have a problem with this, please let me know – I can only act on problems that you tell me about). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU publicity material or otherwise support ACCU, please let me know so they can be added to the list

Computer Manuals (0121 706 6000)
`www.computer-manuals.co.uk`
Holborn Books Ltd (020 7831 0022)
`www.holbornbooks.co.uk`
Blackwell's Bookshop, Oxford (01865 792792)
`blackwells.extra@blackwell.co.uk`
Modern Book Company (020 7402 9176)
`books@mbc.sonnet.co.uk`

An asterisk against the publisher of a book in the book details indicates that Computer Manuals provided the book for review (not the publisher.) N.B. an asterisk after a price indicates that may be a small VAT element to add.

The mysterious number in parentheses that occurs after the price of most books shows the dollar pound conversion rate where known. I consider a rate of 1.48 or better as appropriate (in a context where the true rate hovers around 1.63). I consider any rate below 1.32 as being sufficiently poor to merit complaint to the publisher.

C & C++

C++ from Scratch by Jesse Liberty (0 7897 2079 5), Que, 418 pages+CD @ £27-99 (1.07)
reviewed by Paul F. Johnson

I usually like Jesse's books, but not in this case. I would not recommend this book to anyone as it is just badly written.

According to the user level, the book is aimed at beginners. The book's style for teaching C++ is to program first, explain later, which usually is quite a good idea, but only when what you are doing is explained properly.

From an early stage, things become annoying, as the book seems mixed up. It starts by saying that the code should work on any standard compliant compiler. So why will it not work on gcc 3.3? The book looks like it was written by two people. One person appears to be using a compiler that rejects `iostream.h` and does not use namespaces; the other using a compiler that accepts old, broken code. This is worse than useless for a beginner who needs to type in the code exactly as written and expects it work there and then, without having to worry about why the code does not compile.

Other non-compatible features including a header which is not described for almost 40 pages after it is first used; using old C headers instead of C++ headers – there are quite a few instances of using `time.h` in place of `ctime`, as well as using the C style method of generating a random number (`seed rand()` with `time(NULL)`) instead of the more useful C++ method.

The description of object-orientated programming is rather weak. Even for a beginner.

While the use of the STL is nice to see, the descriptions of the object methods and how they work is poor. This is a common thread. For instance, there is very little on error trapping (essential for the program being developed throughout the book). The index is worse than useless; it does not cover any of the STL methods or in some cases, important sections of the book.

UML is supposed to be covered in the book. If it was, I did not recognise it.

Definitely one to leave on the shelf.

Learning to program in C by N. Kantaris (0-85934-203-4) Bernard Babani Books, 126pp
reviewed by Paul F. Johnson

This is one of those handy to have little books. While it is of limited use to those who are learning the language (it is too short and the material is not adequately described), it is very good to have around for those moments when your brain decides to go on holiday and you just cannot remember how to use `strncpy`!

It is of limited use for those learning as it is way too short to cram what learners require, plus it is not exactly up to speed with respects to the standard (every code example starts with just `main()`).

That said, the questions are just taxing enough to make you scratch your head and the short descriptions of the functions from the main headers are enough to prod you in the correct direction and at seven pounds, it is better than most books at four times the price!

The two strongest chapters in the book cover file operations and string handling with the string handling being far better than the file operations. The rest do the job, but just in no great depth.

An Introduction to GCC by Brian Gough (0-9541617-9-3), Network Theory Limited., 116 @ £12-95 (1.54)
reviewed by Francis Glassborow

There are numerous books purporting to introduce the reader to C or C++. Some of these are actually introducing a specific compiler, indeed some have little if anything to do with learning to program and a great deal to do with a specific IDE. The reason that I mention this is because this book focuses on a set of compiler tools that are generally referred to as GCC. Unfortunately, despite the intentions of the author its title still manages a small amount of mis-direction. It is about a subset of GCC (GNU Compiler Collection) and only deals with using GCC for C and C++ source code.

Please note the use of the word 'Introduction' in the title; it is far from being a comprehensive manual even for the restricted C and C++ parts of GCC. For comprehensive coverage of GCC you will need an up-to-date version of Using GCC by Richard Stallman (published by GNU Press).

Now having warned you about the limited range of the book let me look at what it actually achieves. The book attempts to address the broad needs of newcomers to GCC incorporating the entire range of programming experience. In honesty, I think this is over ambitious and I would not suggest this book to the newcomer to programming, they have too much else to master and too much to learn about computer terminology to feel comfortable with a text at this level. For the rest, this book seems an excellent introduction. I have to say seems because I have had too many years dealing with the technology to be certain that what is clear to me will be equally clear to others.

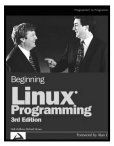
If you can already program in either C or C++ and want to adopt GCC for compiling and linking your source code you have two major choices; you can use a command line environment or you can use some form of IDE. Even when using an IDE you may need to know details of the command line invocations if you need to tweak one or more scripts used by your IDE. You will also find that some details such as those concerning optimisation flags are useful even when using an IDE. While this book does not go into the level of detail that you will find in Using GCC, it does give more than enough details to get you on the right path.

As well as the basics required to use GCC as a command line compiler/linker for C and C++ the book includes a few very brief chapters on such things as platform specific options and how a compiler works. While the former can be useful I find the latter has a smell of stretching limited content to achieve a reasonable page count. Or perhaps the author genuinely thinks this book is appropriate to the raw novice. But if that were the case the language would have to be less technical and the text would need to go into greater detail.

The final criticism I have of this book is that the writing style is ugly. I come from the school of writing that advocates the use of simple English and advises against such things as the extensive use of the passive voice and the third person. Be warned that if you have a similar view you will be irritated by the author's writing style.

Despite my critical comments above I think this book fills a much-needed niche in the marketplace. Those who are less than expert users of GCC for C and C++ will find that having this book on your reference shelf is a cost effective

alternative to having Using GCC to hand. However do not expect too much, it is only an introduction and serious users will probably need more detail.



Beginning Linux Programming by Neil Matthew & Richard Stones (0-7645-4497-7) WROX, 848pages @ £26.99

reviewed by Paul F. Johnson

If you want to learn how to program for the Linux platform, you will not go far wrong with this book. It covers just about everything you will need including SQL, Gnome, GTK+, Qt, KDE, the developers tool chain and how to write shell scripts.

The code is clear and the discussion and explanations are excellent. What makes the book even better is that aspects are not kept in isolation – the material used in the SQL chapter is used further on in the book with the Gnome/GTK+ programming material. This helps bring everything together.

It is assumed that you know how to use gcc and the basics of C programming. This is not a big problem.

The only questionable part of the book is the inclusion of a chapter on writing code for device drivers, i.e. interacting directly with the kernel. Not that this a weak chapter, but having such a chapter in a beginners book is perhaps not that good an idea. Despite that, the code is fine and the methodology behind the chapter is clear.

This is an excellent book for anyone wanting to get involved with the phenomenon that is Linux. Highly Recommended

Java & C#

Java 2 Weekend Crash Course by Sanchez & Canton (0 7645 4768 2), IDG Books, 427 pages+CD @ £15-99 (1.25)

reviewed by Richard Lee

The title comes from the way the book is organized; 30 chapters which should take half an hour each, loosely grouped into 6 sections. If you are following the course guide, this starts Friday evening and finished Sunday afternoon. These contain a run-of-the-mill how to program course with the text condensed down to fit the given time frame.

The book does achieve its aim of teaching the basics of Java programming in a weekend and there are a couple of nice touches but that is about the best that can be said.

You do not get to write your first Java program until chapter 3, the ubiquitous Hello World example, and you have to wait until chapter 6 before encountering anything more to type. The rest is taken up with theory more at home in an A-level course.

When the programming does start it earnest it follows a formulaic approach that could have come straight from a book on programming in C. It is chapter 11, a third of the way into the book, before object-oriented programming is introduced. Recursion and abstract data structures are briefly mentioned but in this condensed form, a beginner is unlikely to understand much and an experienced programmer likely to be insulted by the noddy descriptions.

With so much of the book taken up on introducing the basic elements of the language and other chapters wasted, there is little room left for the actual Java programs and the examples. AWT, the most basic form of creating a GUI in Java, is squeezed into two chapters with just one example. Graphic coding gets a single chapter.

More important than what is in is what has been left out. There is nothing on applet programming, networking or SWING. The book fails to develop a single worthwhile application.

Anyone who already knows a different programming language will find the way the language is introduced tiresome. I also cannot see how someone who has not programmed before is going to learn from this book. Not recommended.

Java and XML by Paul Whithead et al (0 7645 3683 4), Wiley, 309 pages+CD @ £20-99 (1.29)

reviewed by Silvia de Beer

The book uses a special visual layout. All sections are laid out on two opposite pages. The lower half of the pages is used for the screenshots. However, the topic of this book is not very well suited for screenshots: mainly Notepad editors and Command Prompt windows. The screenshots show the example code, and the output of the example Java programs. The examples are showing the use of the various APIs and the concepts explained in the upper half of the pages.

Up until page 65, I considered throwing this book into the bin, if I did not have to write a review: a very poor book. It explains the Java language in a very poor way. I consider this introduction a waste of time because if you do not know Java, you would not be able to learn to program in Java from it. The book even tries to introduce a few OO concepts, it tries to explain what a class is, but on the other hand, it does not even mention the concept of an interface. The introductions on Java and XML contain too many statements that are very debatable or incorrect. After page 65, which explains XML, the book is a bit more useful, but not complete enough. It covers a little bit of XML, DTD, the SAX API, DOM, JDOM, JAXP and even less of Schemas. The appendixes, references on Java and XML, from page 276 onwards are useless, as they are very incomplete. The book fails because it is incomplete.

It almost seems like the authors are themselves beginners and have not really programmed in Java. They talk about copying the various `.jar` files into the Java sdk installation path, to avoid setting a classpath. The book also advises to use the `set` command in the file `autoexec.bat` to set a classpath. I wonder which operating system they are using! The only value of the book is the CD, which contains the Java SDK and the Xerces parser. Of little value of course, because this is all open source, but handy if you want to avoid downloading them yourself.

Java Programming for the Absolute Beginner by Joseph P Russell (0 7615 3522 5), Prima Tech, 502 pages+CD @ £21-99 (1.36)

reviewed by Greg Billington

When this book says it is aimed at the absolute beginner it means beginner to programming rather than a programmer with no experience of Java. That makes it even worse. Making the topic interesting by gradually building up examples that are games is a good concept but the book does not execute the concept very well.

In practice the book launches into Java and programming in too complex a fashion for the total beginner, there are lots of abbreviations (often not explained) and it discusses terms and concepts that are not explained until a much later and do not need to be introduced this early. It seems odd to be using terms like how many bits a data type has without explaining the term, particularly considering the audience of this book. References to hex and octal are not explained and as you go into chapter 3 it covers methods of the random and math class before covering how to use "`if`" and even what classes are. The flow and structure of this book does not feel right; covering `try`/`catch` and the basics of exception handling on page 39 of a 500 page book seems a tad early for the total novice.

The general jokey examples e.g. snippets of Metallica lyrics and how to add comments around them, a fortune teller routine demonstrating random numbers that prints text such as "You will talk to someone who has bad breath" doesn't give this book much of a professional sheen. It may well attract young kids who want to write (or hack) games on their PC but I cannot see it being interesting to anyone else.

On the positive side, it visually looks nice: good font, nicely shaded and laid out tables, screenshots etc and the included CD ROM has the Java SDK and the source code for the games.

Overall not recommended.

Mastering Jakarta Struts by James Goodwill (0-471-21302-0), Wiley, 335 @ £27-95 (1.43)

reviewed by Silvia de Beer

This is a guide to making your first steps using the Jakarta Struts framework. It is a typical example that the average shelf life of computing books is not very high, as the technology of new frameworks like Struts evolve too quickly. The Struts project was created in May 2000, the book published in 2002, and I am reading the book in April 2004. Looking on the Struts website I notice that a lot of things are not described at all in the book: the `ValidatorForm`, the `Tiles` extension, difference between MVC Model 1 and Model 2.

The book contains some curious cut and paste errors, e.g. Part I in the contents has the title "JXTA Overview", it seems like the publisher has used one of its other books as a template.

The book contains three parts; the first part forms an overview of Tomcat, JSPs and Struts. The second part, titled Core Struts, works through a basic example of how to use Struts. The third part of the book is not very useful: it covers the `struts-config.xml` file, and four tag libraries (the bean tag library, the html tag library, the logic tag library and the template library). They are not treated in a useful way; they repeat the attributes of similar tags, often related to equivalent HTTP headers or HTML tag attributes.

People with relatively little experience in writing JSPs and using Tomcat, and who want to read an introduction to Jakarta Struts will still benefit from reading this book. The book is pleasantly written, and guides you by the hand

with small understandable and practical examples.

It is a pity that everything is kept so simple, and more difficult questions are avoided. For example, a Plugin is loading a properties file with the code:

```
File file = new File("PATH TO
PROPERTIES FILE");
```

This is an excellent opportunity missed to treat the general problem of how to load a properties file in a web server's context.

The Tomcat deployment examples are also too simplistic; the book does not even mention how to deploy a web service using a `.war` file, or how to use ant or any other tool to compile your Java files.

Games Programming

3D Games: Animation and Advanced Real-time Rendering by Alan Watt, Fabio Policarpo (0 201 78706 7), Addison-Wesley*, 550 pages+CD @ £39-99 (1.60)

reviewed by Dáire Stockdale

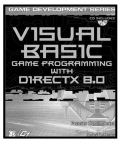
This book is the second volume of a pair of books Alan Watt and Fabio Policarpo on the subject of 3D games. This volume covers topics such as modern hardware accelerated real time lighting, character animation and some collision detection and resolution. It also discusses in depth the 'Fly3D' engine, a proprietary engine developed by Brazilian software company Paralelo.

My biggest gripe with this book is that it is tied to the Fly3D game engine. I felt throughout the book that I was reading a manual for a particular game engine, as opposed to a general discussion of the subject. In fairness to the book, it does mention on the back cover that "the treatment…is built around a specific games system, Fly 3D SDK 2.0". The book comes with a CD which contains the SDK in question, and I believe it is free for non commercial use. However I have to be suspicious of a book that promotes a commercial product without advertising itself as such.

As the book uses the Fly3D engine as its frame of reference, I found this limited the scope of discussion on many of the topics. Instead of explaining the pros and cons of different approaches used by modern games engines to solve problems in this field, as is the case with Eberly's "3D Game Engine Design" book, we are presented only with the method used by the Fly3D engine. This also led to a tendency to present methods as being definitive, when in fact the designer of a game engine system might be better served by different approaches. An example is the engine's use of the Binary Space Partitioning system to accelerate various engine systems. Perhaps in the first volume this system and its merits are discussed and compared with other methods of spatial partitioning; however in this book its use forms the basis of several chapters of discussion, all with relation to the Fly3D engine, without mention of alternatives.

Where the book was not discussing the Fly3D engine, and instead focused on specific topics such as lighting equations, it became much stronger and much more interesting and enjoyable for me to read. Techniques that are only just beginning to leak into commercially available games; such as BRDF lighting and per-pixel lighting are covered clearly and well, and were very informative. These are the best chapters of the book, where it becomes a technical and enlightening read about often-difficult subjects. I suspect that these chapters were authored by one person, and the chapters concentrating on the Fly3D engine and pseudo code by another.

In summary, if you are interested in learning about how 3D engines work, and are happy to use the Fly3D engine as a learning tool, or perhaps if you are considering using the Fly3D engine commercially, then this book will be useful to you. Good as the more generic and technical chapters are, I am not sure they justify a purchase of this book. Perhaps the authors would consider revising the two volumes of this series, and release one that contains only the (excellent) generic discussions, and another volume that focuses on the Fly3D engine?



Microsoft Visual Basic Game Programming with DirectX by Jonathan Harbour (1 931841 25 X), Premier Press, 1100 pages+CD @ £43-99 (1.36)

reviewed by Paul F. Johnson

This is one very large book, which is well written by an author who knows his stuff about games programming. Now, I'm not a fan of Visual Basic (I'm not a fan of its object model or how it works), but DirectX I do like; it is one fantastic library – I honestly wish there was an open source version.

The book clearly explains DirectX and how to use it with VB in very logical sections. Normally, sectioning things is not a good idea, but with something as extensive as DirectX and how you have to write a game, it had to be that way.

Unfortunately, the book is now 2 years old and as such, only covers DirectX version 8. While it's perfectly usable, the advantages of DirectX 9 over version 8 do make games development a lot easier. That is not a fault of the book though, more a problem with the pace of software development.

One of the biggest plusses is that the book also comes with some complete games towards the end and the development process is well defined in the book. It is good to be able to see how the games work and develop; it certainly helps the learner when they don't have some little code snippet which doesn't really help explain very much and in isolation, doesn't help with the understanding.

The book also covers object programming, network gaming, advanced techniques – in fact, just about everything you could want to know to develop your own games under Visual Basic.

The CD that comes with the book contains plenty of demos as well as the DirectX 8 SDK.

A growing number of universities in the UK are now offering Computer and Video Games courses with a push towards XBox programming. If you intend to go to one of these establishments, then quite possibly, this is one book you should think about packing.

Beginning C++ Game Programming by Michael Dawson (1-59200-205-6) Thomson, 335pages @ £18-99

reviewed by Paul F. Johnson

By the time this review arrives into C Vu, colleges and universities in the UK will only be about a month and a half away from the start of a new academic year and with that, there is a new cohort of students going into the increasingly popular Computer and Video Games courses.

Typically, the first year of these requires the students to learn C++ and unfortunately, the quality of the books often recommended are very poor (such as Teach Yourself C++ in 28 days) - the reason for the recommendations are normally down to a very short amount of time before the next stage of the course.

Up to now, there were few decent books that taught C++ (and the STL) and how to use these within a game. This book fills the void. It is a very good book that covers the required material. It does require you to have to a bit more knowledge than a beginner, but not be an expert. As the book is a 2004 vintage, it is also standard compliant (okay, it explicitly has `return 0` at the end of `main()` which is no longer required, but that is minor).

The book covers the major aspects of C++ (such as inheritance and encapsulation) in a very easy to understand and accessible way. The examples for the games are clear, concise, well documented, and very carefully explained. It uses nothing other the STL to demonstrate how to implement the code, which means that anyone with a new-ish compiler can join in the learning.

Why does it not get the highly recommended rating?

The chapter on pointers really lets the book down. While the explanations and diagrams make it easier to understand than many books, it is still made more complex than is really required. There is also no form of exception handling when `new` is used – a fundamental flaw with no real excuse for why is it not treated. There are too many times that I have seen code from second year students where memory handling goes unchecked. It is a pity this was omitted.

If your offspring is off in September, pack a copy of this book and they will not be sorry. Recommended

Methodology

A Practical Guide to Enterprise Architecture by James McGovern et al. (0 13 141275 2), Prentice Hall, 306 @ £31-99 (1.25)

reviewed by Richard Stones

This book sets out to be about enterprise architecture and has been written by six authors, some of whom are well known in the architecture field. Unfortunately, it lacks the necessary organisation and structure to tie their contributions together in a way that gives the reader any real insight into the practice of enterprise architecture.

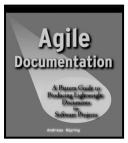
I expected the book to adopt some overarching approach to enterprise architecture, to start with an explanation of this approach and then to set each of the topics in that context. However, after a brief exposition of a framework in the preface, it dives straight into Chapter 1 on systems architecture, which jumps between levels, having sections on detailed technical subjects such as TCP/IP and higher-level themes such architectural types. This is characteristic of the whole book – some chapters are truly about architecture, while others are much more about detailed application development. Most chapters

are about some aspect of architecture: software architecture, service-oriented architecture, data architecture, and so on, but nowhere is the idea of enterprise architecture developed. There is a brief section on the Zachman Framework in the chapter on methodologies, but it is discussed in the context of Extreme Programming, the Capability Maturity Model, Model-Driven Architecture and the Rational Unified Process, not with other candidate enterprise architecture frameworks.

To me, this is a book that has been written by six different people from different perspectives with little attempt to provide any real framework. In the conclusion of the chapter on software architecture the author of that chapter states "Enterprise architecture is in many ways a product of the combined software architectures of the systems in the organization." If that is the case, you have to ask what the need is for the other eleven chapters in the book. Perhaps his co-authors did not tell him.

The book also lacks consistent copy-editing, with stupid errors, like "parishioners" for "practitioners" and the plural of "criterion" spelled "criterions" and "criteria" in the same paragraph.

If there is a theme in the book, it is the idea of Agility and Agile Architecture, referred to in the preface and in some chapters but not others. This subject has potential for an interesting book but, unfortunately, this book is not it.



Agile Documentation by Andreas Ruping (0-470-85617-3), John Wiley & Sons Ltd, 226 @ £22-50 (1.56)
reviewed by Anthony Williams

The title of this book is "Agile Documentation", but almost everything it says is applicable to documentation for any project, whatever methodology is used. Indeed, much of what is said is common sense if you think about it – but how often does anyone really think about it?

Reading this book forces the issue, and hopefully encourages one to think about the purpose, readership and content of documentation a bit more in the future. However, some of the content is particularly important when trying to use an Agile development method, since it contributes to reducing the effort that is wasted on unused or unnecessary (or even unusable) documentation, whilst ensuring that the documentation that is produced is both necessary and sufficient for the project's needs.

The subtitle is "A pattern guide for producing lightweight documents for software projects", which is quite apt. Essentially, the book consists of a set of patterns, divided into 5 groups, each of which describes a particular problem associated with documentation, and some discussion of the solutions. The key points are summarised in what the author calls "thumbnails" – a couple of sentences which appear in bold type in the pattern description, and which are then repeated in the "thumbnails" section at the back of the book. These enable you to browse through the book, reading each pattern heading and the corresponding thumbnail to get an overview of the pattern and determine whether it is applicable for your current situation, or jog your memory.

The patterns are not just presented on their own, they are backed up by experience reports from a number of projects that the author has been involved with. These are used both within the pattern descriptions, and in a separate section at the end of each chapter.

They are not all positive, and are used to highlight the dangers of not following the patterns from the book, as well as the benefits of doing so. Overall, they give the advice a place of reference, and are the source of numerous examples.

One slight issue I had with the book was the number of typos, which was particularly unexpected given the subject matter. However, this did not detract too significantly from my overall impression: Highly Recommended

Agile Project Management by Jim Highsmith (0-321-21977-5), Addison-Wesley, 276 @ £26-99 (1.30)
reviewed by Alan Lenton

After reading the first couple of chapters I was starting to get a little leery of this book, but then suddenly it changed completely, to become one of the best I've read on the subject of project management. The problem was that the book tries to mix advocacy for agile development with hard-headed, well written, how-to-do-it material.

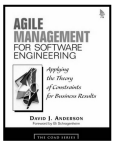
The advocacy stuff is in the first few chapters, and lays it on with a trowel, but once you get through that section of the book, there follows an excellent exposition of agile project management strategies and techniques.

However, this book isn't just for agile practitioners, it also has valuable insights for those who are managing non-agile projects. The book's particular strengths are in its discussions of leadership, and the decision making process. I don't think I've ever seen a decent discussion on decision making before.

The book does deal with the management of large teams, but not in enough detail for my liking, which is unfortunate, because that seems to be where I hear the most criticisms of agile development strategies.

All in all, a useful book, both for project managers and for programmers who have project managers they need to train!

Definitely recommended.



Agile Management for Software Engineering by David J Anderson (0-13-142460-2), Prentice Hall, 312 @ £35-99 (1.25)
reviewed by Jon Steven White

Agile Management for Software Engineering is targeted at managers, team leaders and executives within the IT industry. It sets out to explain how to achieve lower cost, faster delivery, improved quality, and focused alignment within a business.

The first part of the book covers all aspects of Agile Management including production metrics, project management, project planning, resource planning, product management and financial metrics. The author does an excellent job in describing how and why traditional cost accounting systems fall short in software development, and how this can be improved through the application of the Theory of Constraints, a concept originating from the world of manufacturing. Each topic is described very clearly, providing solid background information and real-world discussions to back-up convincing conclusions.

The latter part of the book provides a survey and comparison of a number of software development methods. This part of the book is particularly useful to anybody who needs to manage a change to Agile development, and to choose the most suitable Agile methods for their organisation. Again, the author provides solid information with clear and useful diagrams.

Agile Management for Software Engineering is the best book I have read on Agile software engineering. Writing with clearly extensive knowledge and experience in this area, the author convinces the reader very quickly of the advantages of Agile development. Agile methods have been around long enough now to prove that they do actually work. If you are interested in managing a change to Agile development, curious as to what it can offer, or just want to question the way your organisation currently works, then I'd highly recommend this book.

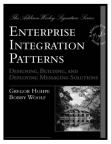
Building J2EE Applications with the Rational Unified Process by Peter Eeles et al. (0-201-79166-8), Addison Wesley, 265 @ £30-99 (1.29)
reviewed by Fazl Rahman

I became sceptical early on reading overflowing praises in the foreword and preface, but my major gripe must be the chapter titles being unnecessarily repeated in full at every reference. Halfway into the book it chafed, though others may disagree. (Typesetting/binding is good though.) Another gripe: I found myself scratching my head a lot at such things "Design subsystems" and "Design packages" being used in the same breath, and needed the appendices often to look up their distinction.

Cutting to the chase: Chapter 2 gives a concise and useful introduction to J2EE. The waffle starts to creep in at Chapter 3 introducing RUP, then I found it progressively harder to pay attention until chapter 6 on Requirements where I liked the material on reviewing and getting user sign-off on requirements 'artifacts' [sic], also something I've rarely seen mentioned in use-case modelling treatments. (The authors provide a nice checklist of items such a review should cover on p 84.)

The section on User Experience Modelling (in Chapter 7) is a gem – an insightful exploration of the GUI within a Use Case based UML model, going beyond just labelling GUI classes with the <boundary class> UML stereotype.

Overall though, after the glowing enthusiasm in the forewords etc (by more than one person) I felt disappointed. Frankly I had to force myself to revisit this book. If I was working on a J2EE project using RUP, I'd probably be happy spending under £25 and (force myself to) read it again.



Enterprise Integration Patterns by Gregor Hohpe & Bobby Woolf (0 321 20068 3), Addison-Wesley, 685 @ £34-99 (1.29)
reviewed by Richard Stones

Enterprise Application Integration is big business these days. Many companies are finding that their businesses demand more "joined up thinking" and a more agile approach to changing business environment. Previously unconnected systems have to be connected and new interfaces developed. Completely new applications, for example selling and claiming on insurance

policies over the Internet, are being demanded of legacy mainframe, batch-mode IT systems. Software vendors are keen to market solutions to these integration problems. EAI tools that are able to interconnect many types of applications and systems, that can integrate web services and form part of a service oriented architecture are fast becoming the centre of an integrated business.

Enterprise Integration Patterns is an attempt to formally describe EAI functionality, concentrating almost wholly, and unapologetically, on messaging as the basis for integration.

The book appears on the surface to be jumping on a "pattern bandwagon". There are many books that claim patterns can be used in a wide variety of different fields of endeavour, and in my experience, few really deliver. However, Enterprise Integration Patterns makes good use of the pattern paradigm to describe how messaging can be used as the basis of an integration platform.

Some of the 60 or so patterns described in the book are fairly simple. For example, the publish/subscribe channel will be familiar with anyone that has used an EAI or messaging middleware. Here though, the patterns are used to give common names and notations for features that have different nomenclature in products from different software vendors. A cross-reference of the patterns to commercial product features is sadly absent. The pattern catalogue is available online at

`www.EnterpriseIntegrationPatterns.com`

Anyone faced with the task of integrating applications in their business using a messaging model or EAI tool will benefit from this book. It succinctly describes many EAI features as patterns, pointing out how and where these features can be used. There are also some worked examples implemented with several technologies such as JMS and Web Services, and mainstream EAI products from vendors including TIBCO and Microsoft. Having said this, the book does aim to stay vendor neutral, and in this is largely successful. If you are doing EAI, recommended.

Software Architect Bootcamp 2ed by Thomas J. Mowbray and Raphael Malveau (0 13 141227 2), Prentice Hall*, 350 @ £39-99 (1.25) reviewed by Emma Willis

This book has been written as a guidebook for anyone thinking of venturing into the world of software architecture from either management or development backgrounds. As you would expect from any good book, it has recently been updated to address changes in the technologies and processes that it examines; this includes a refocus on enterprise technology frameworks e.g. .Net and Java, and a brief introduction to emerging technologies such as Web Services and other XML technologies.

The running theme throughout the book is that of the Army 'bootcamp'. Each chapter has a name and an introduction that lamely tries to tie the chapter's content to the Army theme. I wasn't that impressed!

As a junior developer, I felt that there were areas of the book addressing my desire to rise in the development ranks – providing me with direction, inspiration and lessons to learn in software architecture that could make me stand out from the 'mass' of developers. Additionally, I found some areas of the book to be targeted at those with many more years of experience – perhaps those that had already started their steps into Software Architecture but who need direction, or perhaps need help in addressing problems that they have experiences along the way.

The book includes an introduction to enterprise technologies such as OO-Programming, delving into .Net, Java and particularly CORBA; then explains where these technologies and tools could usefully be deployed. There is also an introduction to design patterns, to software engineering practices, to people management, documentation and communication management and, perhaps the crux of the book – to software architecture lesson-learning and decision-making.

This book is packed full of diagrams and textual examples. Each exercise at the end of the chapter contains an anecdote from the authors. Towards the back you will find an appendix of UML, software engineering and software architecture titbits for future reference.

I loved this book. I shall keep it with me and aim within 5 years to be in the great places it suggests I can be.

Software Development for Small Teams: A RUP-Centric Approach by Gary Pollice et al (0-321-19950-2), Addison Wesley, 272 @ £30-99 (1.29) reviewed by Giles Moran

This book follows the progress of a voluntary software project from start to end. The team performed all of the work in their spare time and were geographically dispersed adding to the need for a process. The team was already familiar with the RUP or Rational Unified Process and had access to some of the Rational tools at their everyday places of work.

The book focuses on all aspects of the projects from the initial construction of the team all the way through to the final post-mortem. The book follows the RUP phases with each phase lasting a number of chapters.

The initial chapters introduce the team and give an overview of the project and the process. The project gets started by chapter 4 where the team discusses various aspects of the project such as milestones and communication methods. The standard RUP phases then follow: inception, elaboration, construction and transition, each as a chapter. These chapters are a good 'by example' introduction to RUP. Implementation details follow the chapters on elaboration and construction. The book ends with a useful project post-mortem (which should be compulsory in all projects IMHO), and a useful appendix.

The RUP is usually criticized for being too large and heavyweight for use by small teams and this book goes some way to addressing this point. The problem is that the team members already know about RUP, and I still think that any team starting out is still going to find RUP daunting.

I liked this book; it is highly readable and entertaining. I learned a lot about RUP and found myself in agreement with the authors about a lot of the points raised. The constant RUP references did annoy, as it wouldn't have taken much to generalise the book for a larger audience. For instance, when discussing version control, ClearCase is used even though the authors agree that it's far too heavyweight for the project. A small discussion on other source control systems such as CVS would have been useful. I know that the subtitle of the book is "A RUP-Centric Approach", but it wouldn't have taken much to make this a more general "Iterative-Centric Approach".

The problem with recommending this book is its scope. Small teams embarking on RUP would gain something by reading it, so for this particular subset of the readership I'd recommend it.

I wanted to read this book as I work in a small team and have wanted to get some ideas on how to improve the development process. I've gained some useful ideas from this book and will encourage other members of my team to read it.

Software Fortresses : Modeling Enterprise Architectures by Roger Sessions (0-321-16608-6), Addison-Wesley*, 277 @ £26-99 (1.30) reviewed by Giovanni Asproni

An introduction to a new methodology for modelling enterprise software architectures: the Software Fortress Model. As the name suggests, the primary goal of this method is the development of systems that are both secure and reliable – arguably the two most important characteristics an enterprise system should have.

The book is aimed at anybody working in a large corporate organization that has a stake on its IT architecture, including developers, architects, technical managers, and the end users.

The Software Fortress Model has two main building blocks. The first the Software Fortress – a "conglomerate of software systems serving a common purpose" that "work together in a tight trust relationship to provide consistent and meaningful functionality to a hostile outside world". The second the Software Fortress Architecture – an "enterprise architecture consisting of a series of self-contained, mutually suspicious, marginally cooperating software fortresses interacting through carefully crafted and meticulously managed treaty relationships."

The methodology also includes a graphical notation, derived from UML, and an adaptation of the Class Responsibility Cards called Fortress Ally Responsibility cards (FAR).

The book is well written and informative: it is readable, contains several interesting ideas, and does not require a strong technical background to be understood.

The book is let down by only two things.

First, it lacks a bibliography section. In my opinion, this is a major problem. New ideas are seldom developed in a vacuum – the author himself admits that many of the ideas in the book are not really new – and knowing what influenced their development can be useful for understanding them better.

Second, some opinions are misrepresented as facts. For example, the comparison of the costs of choosing .NET versus J2EE in paragraph 15.5 is based more on gut feelings than on evidence.

That said, even with its weaknesses, I think that this book is a valuable read for anyone interested in enterprise software architectures, also for people working in small organizations – even if the author claims that the Software Fortress Model is "overkill" for them.

Recommended.

The Rational Unified Process Made Easy by Per Kroll & Philippe Kruchten (0-321-16609-4), Addison-Wesley, 416 @ £30-99 (1.29) reviewed by Giles Moran

The book starts with a quick overview of the RUP (the Rational Unified Process) and some of the Rational tools. It may be best to skip this chapter as it is mostly marketing material for Rational products. Chapter Two describes the philosophy of the RUP by stating the basic principles. They seem fairly sensible and cover most of the items lists in the Agile Manifesto. The RUP is then compared with a number of other software processes to explain how RUP can and is used. A number of example projects are the introduced ranging from a one man one week project to a large distributed project run over two continents. The one-man project is then expanded in the next chapter to very clearly illustrate all of the points raised so far.

Part Two of the book examines the life cycle of a RUP project in more detail. The four phases of a RUP project (inception, elaboration, construction and transition) area covered by separate chapters again using the example projects to aid comprehension.

Part Three is concerned with how to adopt the RUP within an organisation. This starts with a chapter on how to initially configure the RUP. As this chapter deals mainly with RUP software and tools, it was not that useful, as I do not have access to them. The next chapter (Chapter Eleven) is more useful and offers practical advice on how to actually adopt the process. All of the advice seems very sensible, adopt a little bit first, and perform a pilot project to evaluate what parts of RUP are required, all sensible stuff. Chapter Twelve deals with how to adopt an iterative project and is followed by a chapter of RUP anti-patterns.

The final part of the book offers a view on how RUP affects project managers, analysts, developers and testers; each role is the subject of a chapter. A good glossary (required for all the TLAs) and references then complete the book.

"The Rational Unified Process Made Easy" is subtitled as a practitioners guide and in essence that is exactly what it is. It offers a clear and concise introduction to the RUP and the toolset, augmented with good advice and examples. This book is suited to a developer/analyst or manager who will be using the RUP in the near future.

Due to lack of space not all book reviews could be printed in this issue. Reviews of the following books can be found on the website (`www.accu.org`*) and will be printed in the next issue if space permits.*

Fast Track UML 2.0 by Kendall Scott (1-59059-320-0), APress, 173 pages $24.99 reviewed by Derek Graham

The Object-Oriented Development Process by Tom Rowlett (0 13 030621 5), Prentice Hall, 421 @ £43-99 (1.25) reviewed by James Roberts

UML Xtra-Light by Milan Kratochvil & Barry McGibbon (0 521 89242 2), CUP, 106 @ £15-99 (1.31) reviewed by James Roberts

Database Topics

Mastering Data Warehouse Design by Claudia Imhoff et al. (0-471-32421-3), Wiley, 438 @ £31-50 (1.43) reviewed by Richard Stones

Practical RDF by Shelley Powers (0-596-00263-7), O'Reilly, 329 @ £28-50 (1.40) reviewed by Ivan Uemlianin

The Definitive Guide to MySQL 2ed by Michael Kofler (1-59059-144-5), Apress, 802 @ £35-50 (1.41) reviewed by Christopher Hill

Computer Theory

Basic Category Theory for Computer Scientists by Benjamin C. Pierce (0-262-66071-7), MIT, 100 @ £14-99 (1.53) reviewed by Francis Glassborow

User Interface

User Interfaces in C#: Windows Forms and Custom Controls by Matthew MacDonald (1-59059-045-7), Apress, 586 @ £35-50 (1.41) reviewed by Andrew Murphy

Interaction Design for Problem Solving by Barbara Mirel (1-55860-831-1), Morgan Kaufmann, 397 @ £29-99 (1.50) reviewed by Francis Glassborow

User Interface Design by Jenny Le Peuple & Robert Scane (1 903337 194), Crucial, 128 @ £12-00 (1.41) reviewed by Francis Glassborow

Visual Programming by David J. Leigh (1 903337 11 9), Crucial, 142 @ £12-00 (1.41) reviewed by Francis Glassborow

The Web

Web Design Tools & Techniques by Peter Kentie (0 201 71712 3), Peachpit Press, 436 @ £29-99 (1.33) reviewed by Christopher Hill

More Eric Meyer on CSS by Eric A. Meyer (1-7357-1425-8), New Riders*, 270 @ £34-99 (1.29) reviewed by Francis Glassborow

Web Caching by Duane Wessels (1 56592 536 X), O'Reilly, 300 @ £28-50 (1.40) reviewed by Christopher Hill

Leisure

Dancing Barefoot by Wil Wheaton (0-596-00674-8), O'Reilly, 115 @ £9-95 (1.50) reviewed by Francis Glassborow