Contents

Reports & Opinions

Dialogue

Features

Reviews

Copy Dates

C Vu 14.5: September 7th Please try to send submissions in on a timely basis, late submissions are better than none but...

C Vu 14.6: November 7th

Contact Information:

Editorial: James Dennett
76 Lawn Road,
Bristol, BS16 5BB
0117 9653875
editor@accu.org

Advertising: Peter Goodliffe
4 Malvern Road
Cherry Hinton
Cambridge, CB1 9LD
01223 518579
ads@accu.org

The Treasurer: Bryan Scattergood
2 Primary Court
Cambridge, CB4 1NB
01223 357134(hm)
treasurer@accu.org

ACCU Chair: Alan Griffiths
alan@octopull.demon.co.uk
chair@accu.org

The Secretary: Alan Bellingham
020 8998 6964
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): £15

Full (C Vu and Overload): £25

Corporate: £80

Students: half normal rate

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

There are 6 journals of each type 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

++this->change;

Another week, another issue of C Vu, another weak C++ joke. Well, not just another week, more like another two months – but it seems to me to come around more often. I am now ably assisted by Pippa, our new production editor. That this issue appears as good as it does is down to Pippa cleaning up after me! A number of contributors make an appearance for the first time in this C Vu, alongside some regular names. Much is new.

Inevitably not all changes work out quite so well for the ACCU, and Pete Goodliffe has informed us that he needs to scale back his extensive contributions to the ACCU. In some cases editors over-inflate claims of past contributions in a bid to flatter people, but if there are any of you out there to whom Pete's name is not already familiar, a quick look over past journals or your ACCU handbook should suffice to convince you that Pete has made a very real difference. I for one appreciate Pete's past involvement, as well as his willingness to continue to contribute where possible. Maybe you can register your own appreciation by reading his report and considering stepping up and helping to fill the hole. This time I will spare you the rant about ACCU being powered by its members: read almost any of my past editorials if you want to be ranted at! Thanks for all of your work Pete (and please don't be offended that I've declined Word's invitation to change your surname to "Godlike"!).

A new direction for the ACCU is the formation of a Python SIG. We really have no reason to pretend that we are just about C and C++. ACCU is about professionalism in programming, and that means choosing appropriate tools, not sticking to the Swiss-army chainsaw that is C++. (With apologies to Perl fans, who "know" that Perl is the real chainsaw.) This C Vu sees the first article from the Python SIG, an introduction to the SIG and to Python. Expect to see more about Python in coming issues – and if your interest is serious, who not look at the Python projects springing up under the umbrella of the accu-mentored-developers structure?

Next, two requests for some input from you. Firstly, Addison Wesley have asked for your input to a survey; see the letters section for more details. I would suggest that your involvement may well help. As publishers AW have given us many fine books on C++, and a few inexplicable turkeys. In keeping with the history of C Vu editorials as being entirely neutral, I will not single out the contributions of Till Jeske and Jeff Savage as being expensive fire-lighters without mentioning that the "C++ In Depth" series shows what AW is capable of. Please do make your opinions known.

The second request for input is from Bjarne Stroustrup. At `http://www.research.att.com/~bs/applications.html` Bjarne is collating an illustrative list of some applications of C++. Rather than playing Chinese Whispers and trying to give you my own explanation of what makes the list, I'll ask you to read the description at that URL. If you think that applications you are developing or have developed would be illuminating to others, consider mailing Bjarne. His e-mail address is on his homepage, reachable from the URL above. (Brief, clear technical descriptions are best; sales hype is likely to find a home in the bit bucket.)

Having spent so much time appealing to you to contribute, I should make amends by doing so myself. Coming up Real Soon Now, expect to see some coverage of the development of a computerised player for the game of Pente, an inspired excuse to play around with simple genetic algorithms in C++. This should be a chance to follow a fairly simple but non-trivial project from its early stages through to a complete system. I may never win the Pente competition in this household, but I have hopes that the program I am writing might.

In the next issue, expect to see the XML series return with an article on DOM from David Nash. Also there will be a response from an author to one of the ACCU's book reviews (they do get noticed!) and... the rest is up to you. Keep me happy, keep those articles coming! A short editorial this time; I am only here to introduce the acts. Now, on with the show.

James

From the Chair

Alan Griffiths `<chair@accu.org>`

I would like to bring to your attention that our current Advertising Officer (Pete Goodliffe) is standing down. In his time in the post Pete has transformed a hit and miss affair into a steady stream of advertising. This may not seem important to you, but it provides revenue to support our activities (once again we have not needed to raise subscription rates), and provides advertisers with an opportunity to reach our members with appropriate material. I'd like to take this opportunity to thank Pete for a job well done: thanks Pete.

You will appreciate that we now need someone to take over the role from Pete. Please consider Pete's account of the work involved elsewhere in the journal. Thanks to Pete's efforts it isn't a demanding role, and much of it is well-organised routine. Here is an opportunity for someone to make a contribution to the ACCU — it could be you.

During the summer the principle focus of committee activity turns to the renewal of membership. Thankfully we have, in David Hodge, a very competent membership secretary and as Chair I have little need to become involved. Please make an effort to renew quickly to reduce the number of reminders David has to send out. Now is also a good time to persuade any colleagues who have been surreptitiously taking advantage of your membership by reading your journals that they too should join.

Finally, this will be the first issue of C Vu to be produced by our new Production Editor (Pippa Hennessy) and marks the final stage in our taking over the journal related work from Francis. This has taken a little longer than we originally planned, so I think we should thank Francis for his support during the transitional period.

Handbook Update

Bryan Scattergood

You should have received your copy of the 2002 handbook with the previous copy of C Vu. It's since been drawn to my attention that there are some minor discrepancies between the membership lists and the indices by country and by postcode. Those are due to a mistake I made extracting the final versions of those lists from an email.

The country section incorrectly lists Trosset-Moreau as a member in France. It omits Schaper from The Netherlands, as well as Clare III and ershman from the list of members in the USA.

The postcode section incorrectly lists members at PR1 4XN and SW4 0QX. It omits CH44 3AX Barrow, CV35 9PT Newman, EX7 9ER Dyson, HA7 4RG Durer, HP19 8YD Cronje, MK15 0BS Farrands, OX14 4AU Booth, PR2 7ER Grimshaw, SW17 9QU Wakely, SW8 1XQ Precedo, and TS14 8LE Kirk.

If you think some other postcodes are present in the index, but missing from the list of members, please remember to check the corporate members as well as the individual members.

Finally, for the curious, the photograph on the cover of the handbook is of Point Reyes in Northern California.

Advertising and Publicity

Pete Goodliffe `<pete@cthree.org>`

Our advertising schedule this year is looking healthy. I don't have much else to say about this, but what follows is really very important for the ACCU...

Unfortunately I am not able to continue in my post as ACCU advertising officer and so will have to stand down. This is due to a growing number of other commitments, the least of which not being the imminent arrival [1] of a new child. I don't have enough time to devote to advertising to do the task justice.

So I'm therefore inviting a volunteer (or, better, volunteers) to step forward and pick up the reins.

What's involved?

- Sorting out our contract adverts, that is the regular contracted advertisers (includes getting material, positioning the ads, liaising with our printers and the publication editor)
- Keeping relationship with existing advertisers
- Fielding new advertisement requests, and sending out information (I have created a media pack to make this task relatively painless)
- Soliciting new advertisers (this has been done in a big push once a year, with some drip feed work over the months too)

How much time do I need to devote?

- The task can expand or contract pretty much; it entirely depends on how much you want to put in to it. Some times of year are busier than others, but you'll never be swamped!
- You need to be able to use the phone every now and again, although it's mostly done via email.

I will carry on giving input to the advertising, shadowing the new person(s) in their role, so you wouldn't be pushed out entirely on your own.

Please seriously consider whether you are able to do this, and get back to me at ads@accu.org.

I'd only just picked this role up at the last ACCU conference AGM, but again will have to step back. Please consider whether you would be able to perform the publicity officer role; it's really not too onerous. I already have a number of leads that can be followed.

If you're interested in this, please email me and ask what's involved.

[1] Imminent as of writing this, anyway.

Membership

David Hodge <membership@accu.org>

We finished up the year at the end of June with 1110 members, 80 more than the previous year. By the time you read this you should have received either an email or a letter reminding you to renew, if you are a corporate member you will receive a pro forma invoice. As always it will save on administration time if you renew before the end of August. If you have not renewed by that time a second reminder will be sent out. Renewal methods are:- on the website with plastic, cheque, or through our German or USA accounts.

If you are in any doubt about how to renew just email membership@accu.org.

Financial Brief

Bryan Scattergood

The provisional accounts for 2001 were presented to the AGM and are shown below. Our financial year matches the calendar year since that means that year end is at a quiet time, midway between our renewal season in the autumn and our AGM in the spring.

We've traditionally operated at a slight surplus, growing our reserves so that our closing balance is between one and two times our turnover. The figures show that we operated at a very slight loss for 2001, but this is an illusion: we had over £3,500 in outstanding advertising invoices at year end. (Failure to chase those invoices in a timely fashion was my fault, and steps have been taken to prevent that happening again. As I write, all advertising invoices for 2002 have already been settled.)

Note also that breaking even in 2001 was a major achievement. We've managed to absorb the jump in production costs for the journals (associated with the end of the Centaur deal) without increasing subscription fees. In large part this is due to excellent work by Pete Goodliffe as advertising officer, and to the support of our regular advertisers.

With more advertising in place for 2002, production costs under control, and a gradual increase in membership, we should return to our usual surplus for 2002.

Provisional ACCU Accounts for 2001

Income

	2001	2000
Advert	£2,492.60	£1,510.00
Books	£450.00	£510.00
Germany	£38.94	—
Interest	£1,192.86	£953.18
Shirts	£257.00	—
Subs	£30,827.63	£26,183.11
Bank	£3,771.50	£3,402.02
Bises	—	£8,440.00
David	£6,483.25	£9,038.50
Worldpay	£20,332.88	£5,302.59
Total	£35,259.03	£29,156.29

Balances

	Current	Reserve	Total
Opening	£13,673.96	£38,168.97	£51,842.93
Closing	£8,925.05	£42,861.83	£51,768.88
		Surplus	-£56.05

Expenditure

	2001	2000
Accounts	—	£439.13
Charges	£1,190.99	£726.98
Bank	£8.00	—
Bises	—	£370.94
Worldpay	£1,182.99	£356.04
Equipment	£129.18	—
Handbook	£1,189.65	£1,337.57
Journal	£28,132.76	£17,571.52
Centaur	—	£5,816.25
Distribution	£5,943.91	£920.66
Printing	£11,988.85	£2,334.61
Production	£10,200.00	£8,500.00
Misc	£23.24	£34.10
DNS	£11.75	—
Other	£11.49	—
Postage	£2,448.14	£2,316.50
Shirts	£1,003.15	—
Software	£537.47	—
Standards	£540.50	£646.26
Tax	£120.00	£141.55
Total	£35,315.08	£23,213.61

Notes

1. Expenditure for C Vu 12.6 ran over into 2001, and the same has happened for C Vu 13.6 and 2002. The above figures therefore represent six issues of the journals.
2. We expected to be close to break even after the end of the Centaur deal. We're close to that, even though advertising invoices of £3,740.00 were outstanding at year end.
3. Income from shirts ordered via the web are included in the subscription figures. I haven't yet had time to break them out separately. The actual listed income is from those sold at the conference.

Standards Report

Lois Goldthwaite

While considering a ballot to approve international standards for the C# programming language and the Common Language Infrastructure of the .NET framework, the British Standards Institution has formed a new panel of experts to work on these subjects. The standards group ECMA have already adopted these standards and then submitted them for fast-track approval through the ISO process. The BSI C# panel met for the first time on July 15, with a second meeting scheduled for December 10. The ground rules for how actively national bodies will be able to participate in ongoing maintenance of these standards are still to be worked out.

The convenor for the C#/CLI panel is ACCU Standards Officer Lois Goldthwaite, also convenor of the C++ and Posix panels. If you are interested in participating in standards development for C, C++, C#, or Posix, please write to standards@accu.org for more information. There is no charge to join a panel, and no reward except the pleasure of deeply technical discussions.

Mentored Developers: SI Units

Mark Easterbrook
<mark@easterbrook.org.uk>

The big news for the SI Units Project this month is: Code.

Both Pete Klier and Simon Watts have supplied code, which I have put on the web page for discussion.

The last couple of months have also seen some interesting conversations about the problem domain and I think all of us have learnt a lot about SI Units and measurements in general.

It's not too late for new participants to sign up, in fact it's a good time as there's not too much catching up to do, and with code to discuss and regular posting the project is starting to get interesting.

The SI Units project is a great place to learn about SI Units, C++ templates, and the bits of a software development apart from the code.

Mentored Developers: Modern C++ Design

Terje Slettebø
<Terje.Slettebo@iu.hio.no>

The Modern C++ Design project was started as a study group for the "Modern C++ Design" book by Andrei Alexandrescu. The book is about policy-based programming, generic programming, and design patterns. The project was started in April 2002, right after the ACCU Spring Conference.

The original plan was to read the book chapter by chapter, much like the "begin-cpp" project. However, this proved to be difficult, due to lack of exercises (or something like them) to move the project forward. Therefore, there was a relatively slow start, with hardly any postings, lately. However, that has changed rather a lot the last days this is written.

[concluded at foot of page 6]

Dialogue

Student Code Critique Competition

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.

Note that 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.

Student Code Critique 16: The Entries

You were asked to write a response to the author of the code below in which you try to identify the causes of his/her difficulties. Of course straightforward bugs had to be dealt with as well.

```
#include<stdio.h>
void count_up(int start, int end, int step) {
  printf("enter the upper limit");
  scanf(%d; &start)
  printf("enter your lower limit");
  scanf(%d; end);
  printf("enter step size");
  scanf(%d; step)
  int x;
  for(x=start; x<=finish;x=x+step) {
    printf("\n %d",x);
  }
}
int main() {
  count_up(1, 10, 1);
  count_up(-10, 10, 2);
  return 0;
}
```

I only had two entries. Well let me rephrase that; I can only find two entries though I have a nasty feeling that I have lost one. In addition I have a late contribution for the previous competition. That one raises the problem of the time it sometimes takes for C Vu and Overload to reach some members. Of course it mostly does not matter. However this is one column where timeliness is important. I will see if we can persuade our webmaster to place each competition item on our website so that everyone can fetch it from there even if their issue of C Vu is late. Which reminds me, it might help if our website listed the current issue numbers for both C Vu and Overload.
Francis

From David Caabeiro <dac@globalmente.com>

[Please note that David is not a native English speaker. I have done some copy editing but only to help his text flow more smoothly. Francis]

There are several things to point out. First I'll discuss the ones I consider most important.

The student seems to have some misunderstanding regarding function parameters, so I'll try to explain some vocabulary.

When calling a function, you pass it a certain number of **arguments**, which are values the function will work with while executing. When a function is finished, you typically get back a return value (for those with some mathematical background, this is somewhat equivalent to a scalar field).

How do you create a function? First you have to introduce an identifier to the compiler. This is called a **declaration**, and its purpose is telling the compiler what the function looks like. This way, it will be able to check it is used correctly [0]. But this is not enough, you still need to code the function itself and get the storage necessary to hold it. This is called a **definition**.

So how do you declare a function? Simple, you just specify the function name, the parameter list type, and the return type. For example, suppose we want a function that given the (x, y, z) coordinates in space it returns the distance from the origin (yes, this is a scalar field). This function could have three parameters (for each coordinate) and return a value (the distance). So we could declare it:

```
double dist(double x, double y, double z);
```

The first keyword is the return value: a `double`. The parameters are enclosed in parentheses after the function name, separated by a comma. The semicolon indicates the end of the statement. The parameter names are optional, so a valid declaration could also be:

```
double dist(double, double, double);
```

But as you may note, sometimes it's not easy to guess what the parameters are used for.

A function definition is similar to a declaration, except that it has a group of statements enclosed in braces. So in our previous example, a possible definition could be something like this:

```
double dist(double x, double y, double z) {
  return sqrt(x * x + y * y + z * z);
}
```

Note that this time you do have to specify an identifier for each parameter, which will have automatic storage duration [1]. Also note that since the braces replace a statement or group of statements, there's no need for a semicolon. [a semi-colon before the opening brace would be wrong because the compiler would treat that as ending a declaration. One after the closing brace would also be a technical error because it does not meet the requirements of the grammar of either C or C++.]

Now we can take a look at the code and point out the most evident mistakes.

First of all, there's a missing closing brace at the end of `main()` (don't know if this was a real mistake or a printing failure). Inside `count_up()`, there's a missing '`;`' at the first and last call to `scanf()`, to indicate the end of a statement. Also, the calls to `scanf()` are incorrect, as its prototype is

```
int scanf(const char * restrict format, ...); //[2]
```

where the variable argument list represents the addresses of the objects to receive the converted input. So a call should have the form:

```
scanf("%d", &var);
```

Note the use of the comma to separate the parameters, and the & operator to pass the address of the object.

There's also some inconsistency between the messages shown and where these values are stored (for example, the upper limit is stored in a

Mentored Developers: Modern C++ Design [continued from page 5]

There has been a discussion lately on the project mailing list, about how to do it, and it appears now that consent has been reached to set up a Sourceforge project for the group, for experimenting. Also, exercises have been suggested. The idea now is to work on exercises, or other projects as people in the group please, and these may be uploaded at the Sourceforge project, using CVS, and all the members have access to do that.

Our mentor, the author himself, Andrei Alexandrescu, has also expressed interest in this, and that this could in fact lead to useful additions, that may end up supplementing Loki, later. It was found best to have this as a separate Sourceforge project, though, from the Loki Sourceforge project.

With this, this project becomes much more like the other projects, in working on a project, such as this. It's also my impression that it's more motivating to work on something, that may actually become useful. The Sourceforge project will also likely get linked from the Loki project, giving free publicity to it, that way.

The project homepage is at `http://www.accu.org/moderncpp/public/`, and new participants are welcome at any time. These are indeed exciting times.

variable called `start`). Finally, there's an identifier that was not declared: `finish`, probably meant to be `end`.

Now let's see the basic misconception the student is making. He correctly declared a function with three parameters. But then he reassigns all of them with the values gotten from the user. This way, the arguments passed in the function call are lost.

So how can we correct this? One way is creating a function that receives three parameters, as the student already did. But our function will just iterate and print out the results:

```
void count_up(int start, int end, int step) {
  int x;
  for(x = start; x <= end; x += step) {
    printf("%d\n", x);
  }
}
```

That was easy. Now comes the tough part. We have to read the input from the user, and **make sure it makes sense**. Note that `count_up()` doesn't make any validity checking. It relies on external code for this (of course this is a personal choice). This is one possible way to get a value from the standard input:

```
printf("Enter the lower limit: ");
fflush(stdout);
if(scanf("%d", &start) < 1)
  return EXIT_FAILURE;
```

Some things to note: first of all, because of buffering, we should make sure the message is flushed for the user to see it. This is the reason the explicit call to `fflush()` is made. Also note that the return value of `scanf()` is checked against invalid input. As we're reading just one value at a time, we check for exactly one value read correctly. In case of failure, why do we return instead of asking the user for another try? One of the problems with `scanf()` is that it is not a simple task to know the status of the stream in case of failure [3].

At last, we should check the step size is a positive number, otherwise, we'd get strange results (e.g. infinite loops in case of zero, etc.)

I also check that the lower limit is at most equal to the upper limit (anyway, if this checking were not present, the loop wouldn't execute)

So how would our program look like?

```
#include <stdio.h>
#include <stdlib.h>

/* Step up and print within the given limits
 * Makes no validity checking              */
void count_up(int start, int end, int step) {
  int x;
  for(x = start; x <= end; x += step) {
    printf("%d\n", x);
  }
}

int main(void) {
  int lower, upper, step_size;
/* Now we'll try to get the three values
 * needed In case of error the program
 * terminates    */
  printf("Enter lower limit: ");
  fflush(stdout);
  if (scanf("%d", &lower) < 1)
    return EXIT_FAILURE;
  printf("Enter upper limit: ");
  fflush(stdout);
  if (scanf("%d", &upper) < 1)
    return EXIT_FAILURE;
  printf("Enter step size: ");
  fflush(stdout);
  if(scanf("%d", &step_size) < 1)
    return EXIT_FAILURE;

/* Before calling count_up(), we have to
 * make sure the values are valid     */
  if(lower > upper) {
    fprintf(stderr, "Invalid limits\n");
    return EXIT_FAILURE;
  }
  if(step_size <= 0) {
    fprintf(stderr,
        "Invalid step size: %d\n", step_size);
    return EXIT_FAILURE;
  }
  count_up(lower, upper, step_size);
  return 0;
}
```

End notes:

[0] See subclause 6.7.5.3 of the ISO-IEC 9899:1999 for more details.
[1] See subclause 6.9.1 of the ISO-IEC 9899:1999 for more details.
[2] See subclause 7.19.6.4 of the ISO-IEC 9899:1999 for more details.
[3] It is generally recommended to use fgets() and strtol() in these cases.

From: Stuart Golodetz <sgolodetz@dial.pipex.com>

Dear Student,

There are a number of issues with your code that need attention, so rather than writing a lengthy introduction, I'll launch straight in and try and explain the problems. The most urgent problems are syntax errors, since these will stop your program compiling: until you fix them, you can't even use your debugger to see where you're going wrong, since you can't even build the program, let alone run it.

The first problem to note is that you have missed a closing brace at the end of your `main` function. Since the function body must be enclosed in braces, omitting the ending brace will cause problems, due to the fact that the compiler thinks everything up to the closing brace is part of the function definition. In this case, it will reach the end of the file without finding a closing brace, which will cause it to choke.

The next problems we need to look at are in the `count_up` function. Your calls to the `scanf` function are rather haphazard and seem to indicate that this is something that you have been taught, but have forgotten. The `scanf` function is declared as follows:

```
int scanf(const char *format, ...);
```

It takes a variable number of arguments, but the first argument tells it how many to expect. Because it needs to read into the variables you specify, you must pass pointers to the variables rather than the variables themselves (unlike C++, C does not have references). This is because parameters are passed by value, meaning that a local copy of them is made on the stack, hence if the `scanf` function took variables as its parameters rather than pointers to them, it would be modifying local copies rather than the variables themselves. Looking at the code you've written, several points can immediately be made:

```
scanf(%d; &start)
```

1) The first parameter is of type `const char *`. The usual method of passing a format parameter is by using a string literal, which needs to be enclosed in speech marks. Hence you need to write `"%d"` rather than simply `%d`.
2) When calling a function, you must separate arguments to it with a comma, not a semi-colon. If you use a semi-colon, the compiler thinks it's the end of the statement and spits out an error. The most likely reason behind this is that you were trying out an example from a book and misread the character (as the visual difference between , and ; in some fonts is negligible). If this is the case, then I can only recommend you look more carefully at the code. Accuracy is absolutely vital when programming.
3) You are missing a semi-colon at the end of the statement. This is probably a simple typographical error, in which case the only solution is to pay more attention when you're typing code.

When all these corrections have been applied, the final version of the statement is:

```
scanf("%d", &start);
```

Your other `scanf` calls suffer from similar problems, but also an additional one relating to the information about `scanf` I presented above. There is a marked difference (one works, the other causes undefined behaviour: anything from a simple crash to evil demons descending on your house and frying your computer – the standard doesn't specify) between `scanf("%d", &end);` and `scanf("%d", end);`. The first one works as intended, but the second one is a common example of how to crash your program. The point is that if (for example), `end` contains 0, the function will merrily go right ahead and try to dereference it, as if it were a pointer. Needless to say, the results are undefined.

We're entering the home straight as regards syntax errors! Examine the following (whilst bearing in mind that this does not apply if you're using C99):

```
int x;
```

A simple declaration of an integer, you think. Unfortunately, you seem to have forgotten the famous estate agent's mantra: "Location, location, location." In this case, remembering it would have led you directly to the solution to your problem, namely that it needs to go at the start of the scope.

The final syntax error can now be ironed out. In the line:

```
for(x=start; x<=finish;x=x+step) {
```

`finish` is an undeclared identifier. It's fairly clear that you've made this error by forgetting that you called your variable `end`. The best way to avoid these errors in the future is to adopt a consistent naming style so that you know, for example, that given a choice between `end` and `finish`, you always prefer the former. Whilst we're on this line, it's also worth pointing out that it is generally better style to write `x=3x+step;` as `x+=step;`. With all these syntax errors mercifully out of the way, the (now compilable) code looks like this:

```
void count_up(int start, int end, int step) {
  int x;
  printf("enter the upper limit");
  scanf("%d", &start);
  printf("enter your lower limit");
  scanf("%d", &end);
  printf("enter step size");
  scanf("%d", &step);
  for(x=start; x<=end; x+=step) {
    printf("\n %d",x);
  }
}
int main() {
  count_up(1, 10, 1);
  count_up(-10, 10, 2);
  return 0;
}
```

Now would be a good time to rejoice, were it not for the fact that the code still doesn't work as originally intended. To understand why this is so, examine the function calls in `main`. You are passing parameters to the function, yet they are not being used! Clearly this is less than cunning (something technically known as a logic error).

The solution, as it turns out, is fairly simple and has the handy side-effect of fixing the logic error of incorrect prompting (unless `step < 0`, `start` is the lower limit, not the upper limit). The solution is, in fact, to simply delete the prompt code. After all, we have already passed the two limits and the `step` value into the function; we don't need to query a poor, overworked user for them! The revised code now looks as follows:

```
void count_up(int start, int end, int step) {
  int x;
  for(x=start; x<=end; x+=step) {
    printf("\n %d",x);
  }
}
```

This is pretty close to what we want, as well as being much simpler than the original. However, there are a couple more problems we need to fix. First of all, consider the following call:

```
count_up(1,10,0);
```

This seemingly innocuous call will put our function into an infinite loop. (Or as one wise programmer once put it: "Oops!") There are two possible solutions to this:

1) We can define a sensible behaviour for our function if `step` is 0. Perhaps it could output the start value once and then return. Perhaps it could just return without outputting a thing. However, we need to consider whether it actually makes sense to pass a step value of 0. Rapidly coming to the conclusion that it doesn't, we proceed to:
2) Abnormally terminate the program if `step` is 0. The easiest way to do this is using the old workhorse, `assert`:

```
#include <assert.h>
void count_up(int start, int end, int step){
  int x;
  assert(step!=0);
  for(x=start; x<=end; x+=step){
    printf("\n %d",x);
  }
}
```

The function is now both correct and safe. However, the name `count_up` is misleading, since the function outputs the numbers. In fact, it is difficult to find a sensible name for it, which suggests that perhaps it isn't an altogether sensible function at all. It would probably be better to simply write the function body (which is essentially just a for loop) into the calling function, especially since C89 does not support `inline`. A further point is that the output might look better if you had written `printf("%d\n",x);`, although this is an aesthetic point which may or may not be helpful.

Hope this helps,

From the Harpist

I wonder what Francis is after. There are quite a few syntax errors scattered through the code. And I suspect that one of those (the lost closing brace of main) was an artifact of the process of laying out the material for printing *(It was. Francis)*. However one thing sits up and shouts at me, the curious case of a function that ignores its arguments and promptly asks the user for new values. That is where I am going to focus most of my attention. Syntax is easy to fix, but that mistake suggests that the student does not yet understand what a function is and how it works.

Names are Vital

Most programming languages are built round the idea that programmers can invent new things and give them names. We need to distinguish three fundamentally different places where names will occur in a program. These are declaration, definition and use.

A declaration is when we introduce a name into our code. We use names for lots of different things. The most important to start with are names of functions and names of variables. Actually we usually combine the declaration and definition of a variable and write things like:

```
int start;
```

which both tells the computer that start is the name of some int storage (the declaration) and instructs the compiler to provide that storage (the definition). If we do not want to define (provide storage) for a variable but just want to tell the compiler that you will provide that storage elsewhere (i.e. we just want a declaration) we have to write:

```
extern int start;
```

Do not worry about that now, because separation of declaration and definition of variables is unusual in C until you get to things like `struct`. But that is another story that we will leave for another day.

Function names are rather different because as we get to writing larger programs we want to be able to split them into many files (which can be reused in other programs). More than that, we want to separate uses of functions from definitions. A function can only be defined once (well think about it, how would you cope with two definitions, if they were different you would have a mess, and if they were the same you are wasting time writing them twice.) It can be used many times – that is a basic reason for writing functions. Everywhere it is used the compiler needs to know what we are talking about even though it only needs to know how to do it once. The former is what we call a declaration. For historical reasons there are two types of function definition in C, the original one, often called a K&R style function definition and the newer (and safer one) that is called a prototype. Most of us only use prototypes these days and so I am not going to say anything about the other form here.

A prototype has 3 parts. The first part is a type (`int`, `double`, `char*` etc.) that tells the compiler what kind of result it is going to get when the function is used (called). The second part is the name that identifies the function. The third part is a comma-separated list of types that is placed in parentheses. That third part tells the compiler what information the function will expect to be included when the function is used. That last part is called a parameter list. We can, but we do not have to, add a name after each type in the parameter list. The compiler does not care, but those names can help the programmer understand what information is needed when the function is used (called).

A pure declaration ends with a semi-colon. If we end it with an opening brace it becomes a definition. Actually it is impossible to write a definition that is not also a declaration. You cannot tell a compiler how to do something without at the same time telling it what you are doing. So now let me relate this to your code:

Inside your `main` you use `count_up` twice. Assuming that the user understands what the name of the function means s/he would expect the program to output something like:

```
1 2 3 4 5 6 7 8 9 10
-10 -8 -6 -4 -2 0 2 4 6 8 10
```

and is likely to be more than a little surprised to receive:

```
Enter upper limit:
```

Even more surprising is that when s/he responds to the three questions with 10, 1, 1 the program exits without any further output. Well that assumes the syntax errors and use of an undeclared variable are corrected. Now let me give you a reworked version:

```
#include<stdio.h>
```

Note that `stdio.h` is largely made up of declarations of things that are provided by a library file that will have been compiled from the definitions of those declarations.

```
void count_up(int start, int end, int step);
```

That is a prototype (function declaration) that tells the compiler that `count_up` when used will need to provide values for three `int`s. It tells the programmer that these three `int`s will be used as a start, end and step size. The declaration also states that there will be no usable return value (the effect of having a `void` return type).

```
int main() {
  count_up(1, 10, 1);
```

And the compiler can prepare code that will connect those three values with space it assumes the definition of `count_up` will provide. The job of making the connection is done by something called a linker (it links uses with definitions, hence the name).

```
  count_up(-10, 10, 2);
  return 0;
}
```

Now either here or in some other file that we compile and give to the linker we must define `count_up()`. Here is a reasonable implementation:

```
#include<stdio.h>
void count_up(int start, int end, int step) {
  int i;
  if(step <= 0) {
    puts("zero step size, output suppressed");
  }
  else {
    for(i=start; i<=end;i += step) {
      printf("\n %d",i);
    }
  }
}
```

Note how I handle inappropriate values for `step`. Later I would want to modify that strategy, but for now that seems the best answer. Trap bad values, issue a message and then ignore them. I do not bother to trap `start>end` because that simply does nothing anyway.

Now, I can hear muttering about why separate the declaration from the definition. The simple answer is that I can easily change my mind about the definition, indeed I can have several different definitions in different files and make my choice when I come to link the code together. Of course there are other reasons for separating out function definitions from declarations but you will learn those from experience.

Student Code Critique 15: Late Submission

From Brett Fishburne <william.fishburne@verizon.net>

Let me say, first, that on all the systems to which I had access the code compiled and ran flawlessly. This puts me at a dramatic disadvantage.

Unfortunately, lint also failed to find any problems that could cause the segmentation fault on any of these systems.

Thus, my analysis may be completely wrong as I cannot test a "corrected" solution and get different results. Given that, please bear with me and do not laugh too hard if I am all wrong.

My first line of thought was to address what the meaning of the "segmentation fault" could be. A segmentation fault is almost always associated with improper use of a pointer. In my case, it is almost always an inadvertently uninitialized pointer.

Since `tmp=*a` did not cause an error, it seems that the bad pointer must be "`b`." I rigorously checked the logic of the reverse subroutine insuring that "`b`" could never be out of bounds:

Minimum possible value of `b`:

(1) `b` is smallest when `i` is largest, since `b` is reduced in size by `i`.
(2) `i` is largest when `i = (strlen(str)/2)-1`.
(3) by (2) the minimum of `b = str[strlen(str) - ((strlen(str)/2) - 1) - 1] = str[strlen(str) - (strlen(str)/2)]`
(4) Since the `strlen(str)/2` is always less than `strlen(str)` for any `strlen > 1`, the minimum value of `b` is always within the string.

Maximum possible value of `b`:

(1) `b` is largest when `i` is smallest, since `b` is reduced in size by `i`.
(2) `i` is smallest when `i=0`.
(3) therefore, the largest value of `b = str[strlen(str)-1]` which is the last character of the string.
(4) Since `b` cannot exceed the length of a string for any `strlen > 1`, the maximum value of `b` is always within the string.

Indeed, `b` is never out of bounds. That doesn't mean, however, that `swap` could not be passed `NULL` pointers, so my next thought was that `swap` should check for `NULL` pointers simply for completeness. It is clear from the analysis, however, that, as written, `swap` will not be called with `NULL` pointers and the problem is not that `b` has been assigned a `NULL` or rogue value. `b` is always within the range of `str`.

This had me stumped for several days. Finally it occurred to me that I was looking in the wrong place. If `str` was out of range, then `b` would be out of range! Of course, then, so, too, would `a` and we know from the error message that `a` was not out of range (otherwise the assignment to `tmp` should have caused a segmentation fault. Of course, there is the possibility that `a` just happened to point to a valid value, but that kind of "luck" doesn't go well with such a structured code critique. This did point out to me, however, that `reverse` should also check for a `NULL` string.

Again I was stumped for several days. At long last the breakthrough came. "cheese" is a string literal! How is this important? Well, for those of you who have the ANSI C standard handy, flip to 6.1.4 which tells you that string literals should NOT be modified. When a string literal is modified, the behaviour is "undefined" and that means that it should not be done. This is the explanation for why some compilers don't have a problem with this code. The "undefined" behaviour is to allow this particular modification. In real life, however, a string literal should be treated as "`const char *`" not "`char *`".

It seems only fair that in the case of this student's code, the student should have received a compiler error that warned of this *(hang about, how could the compiler do that? It needs to do quite a bit of analysis to spot the problem. If* `reverse` *and* `swap` *were in different files the problem gets real nasty real quick. – Francis)*, not just a run-time error. Fairness, however, is rarely the order of the day. Now we know the cause of the segmentation fault and how to resolve it (don't use a string literal!) The question, however, is whether there is a way to prevent someone from calling `reverse` with a string literal. The answer is no.

Since the assignment is to `reverse` the string "in place" it doesn't make sense to rewrite `reverse` so that it passes back a reversed string. A hefty comment would be nice, but that is about the best that can be done in this regard.

Let us now address the structure of the program itself, having resolved the "dirty little error" and with resolution in hand. The first, and most obvious error, is a failure to include `<stdio.h>`. Since `printf` is used, this inclusion is appropriate, thus the headers should be:

```
#include <stdio.h>
#include <string.h>
```

I, personally, hate the concept of "side-effects." `swap` is nothing but one big "side-effect." On the other hand, `swap` does exactly what you would expect it to do, so the "side-effect" isn't so bad. I still prefer for a subroutine to return an error when an error is possible, so I would rewrite `swap` to account for `NULL` pointers as follows:

```
/* Swaps a and b returning 0 when successful.
 * Does nothing and returns -1 or -2 if a or b
 * is a NULL pointer,respectfully */

int swap(char *a, char *b) {
  char tmp;
  if(a == (char *)NULL) return(-1);
  if(b == (char *)NULL) return(-2);
  tmp = *a;
  *a = *b;
  *b = tmp;
  return(0);
}
```

Along the same lines, `reverse` is similar to `swap`. I would, thus, make similar changes:

```
/* Reverses a string in place – don't call
 * with a string literal! Returns 0 when
 * successful, -1 when called with a NULL
 * pointer, and positive on a print error */
int reverse(char *str) {
  char *a, *b;
  int i, err=0;
  if(str == (char *)NULL) return(-1);
  for(i=0; i < strlen(str)/2; ++i) {
    a = &(str[i]);
    b = &(str[strlen(str)-i-1]);
    (void)printf("swapping %c & %c\n",
                 *a, *b);
    if(err=swap(a,b)) {
      if(err == -1) {
        (void)fprintf(stderr,
            "The first character is NULL\n");
      }
      else {
        (void)fprintf(stderr,
            "The second character is NULL\n");
      }
      return(-1);
    }
  }
  return(printf(
      "The reversed string is: %s\n", str));
}
```

The `main` program must be declared as type `int` according to the most recent standards, so that should be added, the string literal needs to be fixed, and a value returned:

```
/* Creates a string and then reverses it in
 * place. Returns 0 on success, negative on a
 * NULL pointer error, and positive on a
 * printing error */
int main() {
  char notliteral[7];
  (void)strncpy(notliteral, "cheese", 6);
  notliteral[6] = '\0';
  return(reverse((char *)notliteral));
}
```

[Well I would have written:

```
char notliteral[] = "cheese"
```

instead of those three lines. Francis]

The return value from `strncpy` is ignored because it is a pointer to `notliteral`. On the other hand, it is always a good idea to be sure that any string is properly terminated, so the last possible character of `notliteral` gets a `NULL` *(I disagree, you should know what your common tools do – Francis).* In this case, the `NULL` character is necessary, because `strncpy` would only copy up to the 6th character (which did not include the implied `NULL` character on the string literal). [*But it would have copied the null terminator had you used seven instead of six. I think the mechanism you used is far too error prone, being packed with magic numbers. Francis*]

I ignore the return codes from `printf` and `fprintf` (hence the `void` cast) most of the time because there is nothing I can do if those commands fail in this case. I can't warn the user (printing already failed) and the whole function of the program is to print the reversed string. If that fails, then the `main` program sends the error on to the user as its return code.

[void casts are not often used these days because they actually add nothing other than tell maintenance programmers that you know you ignored the return value. But how often is ignoring a return value an error? And, by the way, note that the entire printf and scanf family of functions are ones where the side-effects are what we want. – Francis]

Student Code Critique 17

It is time for a little C++ teaser. Identifying the problem should not be too hard but I want rather more than that. I want suggestions on coding idioms that will make the student's life easier. And I will give you a big clue, making a function template a friend of a class template is not a brilliant idea. Remember that all elements of code are up for criticism. Critiques by September 16 to: `francis.glassborow@ntlworld.com` please. What's wrong with the program below?

```
template <int N>
class T {
  public: friend T operator+ (const T&,
                              const T&);
  private data[N + 1];
};
template <int N>
T<N> operator+ (const T<N>& S1,
                const T<N>& S2) {
  return S1;
}
int main() {
  T<64> a, b, c = a + b;
}
```

I can compile it, but I can't link it– the operator isn't found.

The Winner of SCC 16

The editor's choice is: David Caabeiro. Congratulations David! Please email Francis to arrange for your prize.

Building Sculptures out of Holes

by Raffael Stocker

If fifty million people say a foolish thing, it's still a foolish thing. *Bertrand Russell*

I've been working on an embedded system lately. My specific problem was to install an operating system (in my case GNU/Linux) on a device that, though i386-compatible, doesn't have a cdrom or floppy drive. There's no video card, no keyboard interface, just a serial port, an Ethernet and an ISDN interface. It has a 15 MB flash disk that can be programmed via the serial port. The system's BIOS has a remote control feature that is executed on startup and reads 15 MB of binary data for the disk. The only way to install the operating system is to supply a complete image of an already set up and configured system as one file ready to be programmed into the flash disk by the BIOS utility. Take a file, program the disk, toggle power and the system runs. In theory. In practice, this file must come equipped with a properly compiled and installed kernel, a boot loader, a partition table, and so on.

Under Linux, with a few tricks, you can treat a file just like a disk (i.e. a block device). Then, to create the operating system image, use "dd if=/dev/zero of=image bs=512 count=31296" (for this special system) and you get a file called "image" with the desired size consisting only of zeros. You partition it with fdisk, mount it, copy the kernel onto it and install the boot loader (LILO). fdisk and lilo, the LInux LOader map installer, took about 10 seconds each on my machine, dd took another 3 seconds. This was clearly too much, so I decided to speed it up. After experimenting a day or two without promising results, I figured that I had to write my own partitioning tool. This could be kept very simple, since it didn't have to do protocol with a real hard disk. It was just a matter of writing a few bytes at a fixed position in the file. That done, the whole image creation process took about 13 seconds, 10 seconds less than before. This seemed to be the sound barrier because replacing lilo with an own implementation was far to much work. And optimising away the 3 seconds of dd wouldn't pay off, right?

Wrong. I was curious. I had the idea of replacing dd with a small program that would just create the 15 MB file as a file without real content. Even without the zeros. In Unix jargon, this is called a file with holes. You create a new file, fseek to some position p, write just one byte and close it again. What you get is an empty file with p+1 bytes in size. On the disk, however, this file would take almost no space, at least if your file system supports that. With that change, I anticipated a total runtime of about 10 seconds for the image creation. When I tested, it took only about 3 seconds! Clearly, the file with holes has sped up lilo.

Why do I tell you this story? Because I found it just funny how my curiosity about these files with holes led me to this non-obvious optimisation. This was clearly a matter of coincidence. In the last issue of CVu, in his "Professionalism in Programming" column, Pete Goodliffe wrote about curiosity being one of the key qualities of a good programmer. Could it be that this curiosity more often than not leads to such lucky coincidences, such solutions that can't be planned? solutions that can't be designed? Could it be that some of the best features in software out there were found by accident by a curious programmer? It's clear that one doesn't become a good programmer just by having luck. But maybe one gets luck just by being curious.

Raffael Stocker

The Wall

Letters to the Editor

James,

Francis' comments on C# in the "Francis' Scribbles" section were right on the mark.

I think it can be difficult to assess new technology from Microsoft in an objective manner, without getting distracted by issues such as Microsoft's domination of the industry and whether it has pinched ideas from others or not.

I think if Java did not exist and Microsoft came up with .NET most commentators would have hailed it as great.

Of course, it's unlikely that Microsoft would have come up with something quite like .NET without the existence of Java. But, nevertheless, we can still assess .NET on its objective technical merits — ignoring whether it's copied from others or not.

In any case, no technology exists in a vacuum and everything has been influenced by something else to some degree.

C# has clearly been heavily influenced by Java. But on closer examination you can see that it has also been influenced by C++, Visual Basic and Delphi. And it is closer to C++ than Java is to C++.

I think it's been carefully thought out, which is not to say that some things couldn't be better. But that applies to all languages.

Re: Francis' remarks about knowing more than one language, I agree. Even if you don't use more than one language you should definitely study several. I've found that concepts encountered in other languages, such as Eiffel, have enabled me to improve my programming in other languages such as C++. Different languages have different idioms and learning about these, both when they can be transferred and when not, improves our usage of our primary language(s).

Kevin McFarlane <kevin@atech.globalnet.co.uk>

Dear Editor,

I am a freelance C++ developer. As a long-term casualty of the IT slump, who may have been unemployed for over a year by the time this letter is published, I thought I would comment on the so-called "skills shortage." I have made similar points to a number of IT commentators in recent months and I've only had a single (somewhat inadequate) response.

1. With the current downturn there certainly is no skills shortage.
2. Prior to the downturn there was no skills shortage either. There was an experience shortage. There never really is any shortage of workers capable of doing the work that needs to be done. Lots of us become knowledgeable in a number of technical fields simply by reading books and practising. This does not make us experts but it does mean that in many cases we can get up to speed with a new technology in a matter of weeks and often less.
3. Suppose we take the charitable view that there really is a serious skills shortage and that formal training is required. What is almost always overlooked is that employers are almost never interested in workers who have merely been trained. They always want experience. If employers were more relaxed about taking on workers with knowledge in a particular field but no experience – even paying them slightly less for a few months - I think that many workers would happily fund training for themselves. The government would not need to be involved.
4. Note that in the majority of cases we are talking about people who are already experienced programmers but may not have experience of skill X, rather than those who have been trained as programmers but have no commercial experience. Clearly the former are likely to be more effective than the latter.
5. If the government were to invest more in training this would be a waste of taxpayers' money while employer attitudes persist.
6. When employers train their own staff they are less concerned about the fact that the trainees have merely been trained (I assume).
7. A disincentive to employers' investing in training is the prospect of trainees' leaving for better paid jobs elsewhere. Though I suspect this is only possible if the trainee obtains, say, a vendor qualification plus a certain amount of experience in the field.
8. However, a frequently cited reason for workers leaving for better jobs is that their employer wouldn't invest in training. It would be interesting to find out whether companies who invest in training their staff do better overall than companies who don't.
9. Some say companies should recruit on the basis of attitude and talent rather than technical knowledge. Then the successful ones should be trained up. This concurs with the findings of a US report on the skills shortage by Professor Norman Matloff (see "Debunking the Myth of a Desperate Software Labor Shortage" `http://heather.cs.ucdavis.edu/itaa.real.html`, especially section 7) and I agree completely. In fact, in my experience, attitude is more important than raw ability. (Many "clever" programmers are lousy programmers.) So perhaps "talent" needs to be interpreted in a wider sense.
10. However, it is far less effort, initially, to weed out job applicants on the basis of scorecard skills matches, so I don't see this problem being solved anytime soon.

Does anyone have any solutions to these problems?

Kevin McFarlane <kevin@atech.globalnet.co.uk>

Hello James,

In response to: How do I view the programming part of my life?

I originally joined the ACCU as a hobbyist. I had a little programming experience at college (Basic and Fortran), but my true inspiration came to me while at work. My dad owned a grocery store in rural eastern Virginia, and I evolved into a sort of 'system administrator', among other things since I was the only one who could do a little more than turn on a PC. Our business grew and we eventually updated our equipment to two PC's and a front-end-scanning system. The front-end, and the two PC's were all wired together (I guess that would be called a network!) and information was transmitted via modem to and from the warehouse. The desire to learn more about programming surfaced as a result of my trying to improve the performance of data sharing between systems, which required creating and formatting batches of item data and sending it from the 'main' PC to the front-end-PC, to two other applications, and then to the warehouse mainframe. I also wrote batch files, menus and application scripts, to make this hodge-podge of a system easier for our employees to use.

I poked around, and asked questions as to which language would be best to learn. I decided on C++. What a daunting task that turned out to be! I bought several books, learned about objects and classes, polymorphism and inheritance, and discovered the ACCU. I managed to write a couple of console apps that bridged some gaps between the software at work, but I also laid the foundation for the future that I am realizing today.

The county grew, competition moved in, and we went out of business. After twenty years of grocery store employment I have chosen a new path. I took a job close to home as a carpenter, and have returned to school, on-line. I am a computer science major at Mercy College in New York City. I have just completed two semesters of Java, and I am participating in the mentored Java project here at ACCU. The project provides an excellent duality in my development as programmer. Where else can you find a consortium of experts, willing to help you gain experience in this field for free? I don't know of any.

Well there you have it,

JEFF IN A NUTSHELL
A Desktop Quick Reference
Thanks,
Jeff

Quick thought on Ruby

After reading Francis' book review of "The Ruby Way", I thought I'd share a quick thought based on what I've read on a number of websites (although I haven't actually tried Ruby). To compare Ruby with Python, as is so often done, does not really reflect Ruby's design objectives; Ruby was meant to be a successor to Perl. In Japan, Perl and Ruby are often mentioned together, and there is even a "Perl/Ruby conference" in Japanese. Perl and Ruby both have quite complex syntaxes, and people who like the simpler design of Python are for this reason unlikely to like either Perl or Ruby.

I'm sure that Perl and Ruby have their place even if they're not the One True Language (if there is such a thing). I know there are some ACCU members out there who are Perl programmers; perhaps one of them might like to write us a defense of the Perl/Ruby design philosophy.

Silas S Brown

Francis' Scribbles

by Francis Glassborow

General Issues

When I write this column I try to provoke readers into thinking. I do not mind if you completely disagree with me, I do mind if you do not think about what you read. From my perspective the great failure of modern democracy is that the masses abdicate responsibility by handing it over to a small number of individuals. Too often we seem to ignore that individual politicians have their private agendas that may have little if anything to do with the well-being of the wider communities they claim to represent. Those of us with relevant technical knowledge have a duty to criticize (constructively) and in a way that allows others to understand the issues. Readers of C Vu are way above average in their understanding of information technology. I will come back to this later in this column, but for now think about it.

Personal Views

I was playing Bridge a couple of evenings ago when an long term acquaintance who makes a living writing educational software arrived at my table. As we finished the round slightly quicker than most of the room I enquired how things were going. He said fine. In response to my query he added that he was still programming in C++. (Now to give you a little background, when he first converted from Visual Basic to C++ I had asked him why he had chosen to use Visual C++. His response was that his friends/colleagues had told him that was the best choice and everyone used it.)

This time I asked if he wasn't tempted to look at C#. His response was that he did not want to tie himself down to an MS proprietary language. MS were a company driven by their marketing department and what they had done to Java was a disgrace. I asked what books on software development he had read recently. He looked at me as if that was a very rude question and opined that real software developers had enough to do without reading books. There seemed no point in asking him about attending conferences or consulting others via relevant newsgroups.

There is a consistency about his views in that he does not find it necessary to consult with others who may have some genuine input. He works alone and acquires programming skills by the seat of his pants. He is far from being the only one to work like this. His programming is considerably above average and the educational software he produces is much better than average. Neither of those statements says very much other than to highlight the abysmal state that general software development is in and most particularly the pathetic state of too much educational software.

We live in a World where new technology floods us. What would you think of a five year old graphics card, motherboard, hard drive? You might find some use for them but not for much longer. So what do you think about someone whose development skills were last upgraded five years ago? Just as the cutting edge hardware circa 1997 is badly dated today, so is your 1997 skill set. The only reason that so many get away with it is that many others are still using their 1992 skill set. The fact that many readers of this column may think I am exaggerating is indicative of the problem.

In my exit interview from school I was asked if I thought I had learned how to learn. When I said I thought I had, my headmaster said 'Good, we have been successful.' He was right then, and his view of the principle objective of education is even truer today. If you did not learn something new yesterday, the day was wasted. If you did not upgrade your programming skills last year then you have stopped being a programmer and become, if only temporarily, a software hack (in the journalistic sense).

A Book

Some of you know that my first intellectual love was mathematics (indeed I was once caned for reading a book on mathematics when I should have been doing my French prep – homework if you are unfamiliar with the boarding school term). I was in Blackwells the other day browsing through the mathematics section – noting that it had rebounded from its low point five years ago. My eye fell on a book titled 'The Maths Gene'. A quick look convinced me that this was a book I wanted to read, so I bought a copy.

I haven't finished it yet because keeping up with my review reading has got in the way, but even so I know that it is a book that should be widely read. It should be read by students of mathematics so that they understand why others find maths difficult, it should be read by parents so they understand why maths should not be hard and it should be read by political commentators so that they understand why simple arithmetic is easier for a Chinese child than it is for an English or German one.

If you find mathematics (no, not arithmetic, but the science of patterns) easy, read this book to understand why, and why others find it hard. If you find it hard, read this book to discover why you are not alone.

However what struck me as I was reading this morning was how similar the author's description of doing mathematics was to the way I program. First immerse yourself in the problem and the context (he describes it as building a house) and then you will find the solution somehow surfaces. I program the same way. I think all round the problem explore different approaches in my mind and then it seems to gel and I get down to cutting code. I wonder how many of you do something similar.

An Issue of Privacy

Time to get back to politics.

I wonder how you would respond to a law that required the Royal Mail (US Post or whatever) to keep a copy of all your letters, postcards and the contents of all your parcels. How would you respond to a government that required your telco to keep a recording of all your telephone conversations? What about keeping a record of all your journeys?

The reason that this has never been done has little to do with your government being more considerate of your privacy and a great deal to do with feasibility. Suppose that back in 1951 the then government had required recording of all phone conversations and copying of all correspondence. Everyone would have fallen about laughing because it would just have been impossible and everyone would have ignored it as pure political fantasy. Furthermore even if successful, it would be useless because it would have been impossible to extract any information from such a heap of data.

Of course these days it is still impractical to copy every letter but it is far from so when it comes to electronic communication.

The second great defense of our liberty was that even given all the data it used to be practically impossible to mine it for the information you wanted. Stealing the tax records for 1960 would not get the thief very far because the sheer bulk of information would have made it useless. Stealing the tax records for 2001 is both easier and more useful. Once you have it electronically you can start mining it for profitable data.

Look back at your email. Can you? Do you have an archive of all your email? If not how are you going to demonstrate that an apparently incriminating message wasn't once the context of the other messages is taken into account? Even worse, how about the implications of steganography (hiding information as in the low bits of a graphic). Of course no one is silly enough to hide critical information without also encrypting it. Now try this scenario. A search of your email records leads the authorities to think you were involved in some crime they are trying to solve. They now look at all those pictures of your grandchildren you received electronically. Throw a few terra-computers at them and the authorities should be able to find a few hidden messages whether they are there or not.

Face recognition is not far away and then all those CCTV systems become so much more useful.

Identity cards really ought to have some chip technology buried in them so that we no longer have to worry about carrying grubby cash around with us, just let the bus, toll booth etc. recognize you. Swiping the card is such a bore so let the systems do direct interrogation untouched by human hand. Once we can do that, how long before the police point out how useful travel information would be in tracking criminals.

But worse, like tax records, once such data is stored electronically it becomes possible to steal it and search it electronically. Even for the completely law abiding citizen there are many things that they would not want known by their boss, parents etc. And that leaves them open to blackmail.

Am I being paranoid? Probably, but that does not mean they aren't out to get us. The greatest protection of our privacy is the anonymity of the masses. We are close to being able to strip that away. So what do we do about it? I wish I had an answer. Just remember that the proposal for road pricing that surfaced a few months ago (actually I think the idea is a good one) relies on being able to track every vehicle in the country at all times. The problem is that we are not far from being able to do that. That scares me.

Separate Compilation of Templates

Many of you know that there has been a long standing promise that templates will be separately compilable. Some of you may even know that a keyword, `export`, was added to C++ to provide extra support for such, or at least so it was claimed.

However I have recently come to realize that the very way that templates are specified in the C++ Standard means that separate compilation in the sense most of us think of it is impossible for templates.

Non-template classes and functions can be implemented without the compiler having to have any knowledge of the context in which they will be used. That is exactly the purpose of placing implementation code is separate files and providing information necessary for use in header files. We decouple application code from implementation code. One of the weaknesses of using inline functions is that we overrule that decoupling. (Actually as optimizers get better, the need for explicit inlining steadily goes away. There is a mode in the latest release of VC++ that delays code generation till link time and so supports even better optimization if you are willing to pay the price of longer link times)

Now as we currently have templates specified we can NOT decouple the implementation from the point of use. All that export does is to delay the instantiation to some more convenient stage such as a pre-linker. This comes at a heavy cost, both to the user and to the compiler implementor because (potentially a great deal of) context information must be saved at each point of instantiation. `export` removes the need to recompile the user files when a template implementation is changed but that is not sufficient for separate compilation.

Now to my question: do you want genuine separate compilation of templates to be added? By this I mean a mechanism whereby the implementation need know nothing about the context of use. I need to know your feelings on this issue to decide if it is worth putting in the effort to get a suitable change adopted in the next C++ Standard.

Problem 3

As I know that many of you cannot put your hands on the last issue quite as easily as I might like, I have to re-publish the last little problem to give you a chance to understand my comments. So here is what I gave you last time:

My dictionary defines percentile as '*the value below which fall a specified percentage of a large number of statistical units (e.g. scores in an examination).* The quartiles, lower and upper, are the 25th and 75th percentile. The median is the 50th percentile. For the latter statistics has special rules for small samples where it is often necessary to select a representative value that is not a value from the sample.

Now look at the following code for computing a percentile in the form of a supposedly STL conformant algorithm.

```
template<class RandomIterator,
         class ValueType>
ValueType percentile(
   RandomIterator start,
   RandomIterator end,
   ValueType percent)
{
   typedef typename
      std::iterator_traits<RandomIterator>::
         difference_type DifferenceType;
   DifferenceType n=end-start;
   ValueType rank=(n+1)*percent/100;
   DifferenceType
     intRank=static_cast<DifferenceType>(rank);
   ValueType fraction=rank-intRank;
   RandomIterator pos=start+intRank;
   ValueType result;
   result = *(pos-1) * (1.0-fraction)
                  +*pos * fraction;
 return result;
}
```

What is the fundamental flaw? Let me help you:

```
int main(){
   int array[10] = {0,1,2,3,4,5,6,7,8,9};
   cout << percentile(array, array+10, 25);
   cout << percentile(array, array+10, 25.0);
   return 0;
}
```

For those not familiar with the STL the above mumbo jumbo is a fairly typical piece of template code. It starts by declaring a function template with two template parameters. The names are a clue to correct use. The first template type parameter is going to be used as a random iterator and this will be deduced from the first two arguments of a function call. The second type parameter is going to be for a value and will be deduced from the third argument.

The first incantation of the function body says: 'take the first template parameter type and look up in the `std namespace` to find a specialisation of `iterator_traits` for that type. There you will find a `typedef` that gives meaning to `difference_type`. Rename that type `DifferenceType`. The purpose of that is simply to make the subsequent code more readable (but it helps to know the STL naming idioms)

So let us look at the first function call:

```
percentile(array, array+10, 25)
```

The compiler deduces that `RandomIterator` is a synonym for `int*` because that is type of `array`, and confirms that this is correct because that is also the type of `array+10`. It looks up the `difference_type` for `int*` and finds that it is `ptrdiff_t`. It then looks that up and will probably find that it is `int`. So our `DifferenceType` is `int`. Next it uses the third parameter to deduce that `ValueType` is a synonym for `int` (the type of 25). Now let me rewrite that instance of a template function in a non-template form.

```
int percentile(int* start, int* end,
                                int percent) {
   int n=end-start;
   int rank=(n+1)*percent/100;
   int intRank=static_cast<int>(rank);
   int fraction=rank-intRank;
   int* pos=start+intRank;
   int result;
   result = *(pos-1) * (1.0-fraction)
                  +*pos * fraction;
 return result;
}
```

Now do you see the first nasty problem lurking inside that smart template code? Let me be honest, I completely missed it until I manually instantiated the template; writing robust template code is hard work. Have you seen it yet? What on Earth is that magic 1.0 doing in code that has been carefully typed almost everywhere else? And when you notice that, what about that magic 100. Actually the code works but more by chance than by design. Rewrite `(n+1)*percent/100 as percent/100*(n+1)` and it fails.

Now look at the second instantiation of the template. Short-circuiting to save space and we get the following code:

```
double percentile(int* start, int* end,
                          double percent) {
   int n=end-start;
   double rank=(n+1)*percent/100;
   int intRank=static_cast<int>(rank);
   double fraction=rank-intRank;
   int* pos=start+intRank;
   double result;
   result = *(pos-1) * (1.0-fraction)
                  +*pos * fraction;
 return result;
}
```

Now let us try to understand what was going on in the mind of the writer (I have an advantage because I discussed it with him). He wants to select the type of the return value even though we might expect that to be the type of an array element. This is intended to allow for percentiles coming in the interval between specific elements (e.g. the 50th percentile – median – of a list of ten objects will be halfway between the fifth and sixth). However the way he has passed this type is by using a parameter whose type is used internally. The consequence is intermediate values are wrongly calculated in the case where the `ValueType` is `int`. Check it out and you will find that `fraction` is always zero in this case which means that no interpolation is done. That is incorrect, even if you simply want an integer answer.

If you hijack a genuine parameter to manage the return type your code will not be reliable. And if you are still unconvinced, try reworking the code for an array of `float` or `double`.

Problem 4

When doing code inspections you need to cultivate a suspicious mind. In that light consider the following simple function and comment on what you would check and what minimal changes you would require.

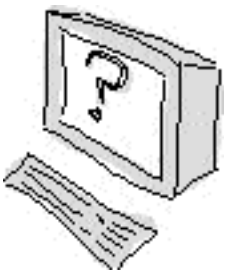
```
void foo(){
   mytype* mt_ptr = new mytype;
   bar(mt_ptr);
   delete mt_ptr;
}
```

Features

Professionalism in Programming #15

The outer limits

by Pete Goodliffe <pete@cthree.org>



I like sweeping generalisations and tenuous metaphors. Sue me. I've also been doing my research. I reckon there are over 40 churches in the city I live in. Now each one of these is subtly different; different types of people go to each, they do different things. They have different concerns and ways of working. They exist in different areas. However they're all doing roughly the same thing (or should be).

What on earth has this got to do with programming? Well by a somewhat tenuous link, software development works in pretty much the same way. Not in that we all file into a building every Sunday morning (well, most of us don't). Perhaps to outsiders we do both appear to engage in bizarre rituals and invoke arcane rites to get our own way with things that are out of the control of normal human beings.

But what I'm really trying to drive at here is that there is no single way to program, no one methodology that solves all problems. There is no one programming language. There are different classes of problems to be solved in many, many different arenas. In this article we'll investigate this, discover some of the common domains we program in, see how they differ, and learn what the particular problems and challenges are for each.

Some of these arenas overlap. That's natural. Nothing's ever quite as clear cut as you'd imagine. The following descriptions are necessarily general, since each of these are big fields with lots of variation within. Nonetheless, this should give you a flavour of what's going on out there.

Applications programming

This is what most non-techies would think of when you mention the word programming. It's probably the broadest category we'll consider here.

It is programming 'applications' – self-contained programs – for single user workstation-like computers. In this world of development the focus is on end-users and how they use their desktop machines. Almost exclusively these days it will be *Win32* programming. Win32 is the Microsoft unified API for its 32 bit operating systems (Windows 95, Windows NT, upwards to Windows XP and whatever other daftly named 32-bit OSes they care to release in the future).

There are many common languages for this kind of work; our familiar C and C++ are in there. We also see common use of Visual Basic and Delphi, plus a number of libraries and frameworks like MFC, and Qt. Although recently you hear a lot about Linux programming, that's still not where the applications work is these days.

Modern application level programming has come on a long way in recent years. We now have rich environments to work in, threading support, good user interface libraries, and facilities for network transparency, for example. There is much more operating system support provided making applications programming easier, but also meaning there's a lot more to learn as you get started.

With all this extra support available we have seen the bar raised on what a 'good' application is. What was acceptable application behaviour a few years ago is not today. People expect standard interfaces and look-and-feel, good responsiveness, and feature packed programs (even if they will only use a fraction of the features available). The huge 'professional' applications marketed today are the result of large development teams, with groups within them specifically focusing on usability issues.

As the times change we are also seeing more of a move towards web-based systems, applications that run on browsers. This cuts into the enterprise or distributed programming arenas (described below) somewhat.

Typical products: Desktop applications like web browsers, spreadsheets, etc.

Target platform: This tends to be the same kind of machine you are doing the development on (more often than not an x86 Windows PC).

Development environment: Normally the same workstation you run the program on. Modern IDEs (Integrated Development Environments) create comfortable working environments for the programmers, bringing the editor, compiler, debugger, and help systems together in a single unified point-and-click interface.

Common problems/challenges: Users expect high quality programs that conform to standard interface principles. More features than anyone knows what do with are order of the day, so much so that new product revisions these days tend to introduce more features (and bugs) than any problems they might solve. This appears to be what the market demands.

Systems software

Figuratively speaking our applications sit on top of rich system libraries. If applications programming receives a lot of support from the underlying system then someone's got to supply an underlying system: this is 'systems programming'.

It is generally for workstation machines too, as above, but not aimed at the end-users so much. This concerns a lot of the low level logic that interacts with the computer at a very basic level, and also middle-level support frameworks that don't interface directly to hardware, but provide important services to the system.

The kinds of work in this arena are: writing device drivers (controlling devices such as printers, storage media, output devices, etc), writing common shared libraries and utilities for managing scarce resources, implementing the actual operating systems controlling the computer, and providing components such as filing systems and network stacks. Even compilers and installation tool suites can come under this heading. Each requires a different approach and presents its own subtle set of challenges.

Typical products: Device drivers, a window manager, or a graphics subsystem.

Target platform: As for applications programming.

Development environment: Often very similar, too. Writing devices drivers and operating system components can tend to screw with the computer and make your system unstable, so it's not uncommon to develop on one machine, and run the code on a second system. C is by far the most common language in this arena, although some library level work is done in C++ these days.

Common problems/challenges: The key issue to deal with for all these system components is they must be *very* stable, since they are foundational blocks of the entire computing environment. Whilst an application might crash and have a chance to save work and gracefully recover, a device driver rarely has such a luxury; it is required to work all the time that it is running. This could be an awfully long time, so even small memory leaks will be a major problem. The code will be required to be efficient, both in terms of space and speed, and will need to be appropriately tailored to the particular operating environment.

Embedded programming

Computer technology is showing up everywhere in our daily lives, whether we're aware of it or not. Consumer electronics products are requiring more and more software for control, operation and management. More often than not this software is invisible to the device's user.

Embedded software developers work under tight constraints. A user interface (in the commonly understood sense) may not exist; in fact there may be no direct interaction with a user at all. Usually there's not much memory, and the code is running on slow processors. Embedded systems are designed to do one job, and to do it reliably. Consumer electronics is built down to the lowest price possible, meaning there isn't a lot of bells and whistles available to use, and you have to shoehorn a lot of software into the available device space.

It should appear as if the software is not there, the embedded device should just work. All the time. Failure is rarely an option in this kind of work. Contrast this to a desktop computer – it's a general-purpose machine. It has to be able to word process, play movies, browse websites, read email, do your accounts, etc. As users we've been conditioned to accept the odd crash and bit of instability here and there. Embedded work is a totally different ballpark.

A good example of this is the modern car industry. We see vehicles manufactured with several embedded systems, controlling all sorts of things. However, the users (in this case the driver and/or passengers) don't have to be at all aware that there are any microprocessors whirring away under the bonnet. They expect the car to just work. When an engine management system fails, the users become acutely aware of the software! Think also about mobile phones. They are obviously computer driven devices, but not many consumers think of them as a computer.

An embedded system is basically a combination of a small computer and either a real-time operating system, or a simple controlling program. It will have direct control over the hardware on the device. Embedded systems are usually developed for specific purposes. In general, embedded systems have only one piece of software running on them, no highly complex threaded programming environments are used.

You might consider that programming applications for hand held devices like PDAs is embedded-level or applications-level work, depending on where you stand.

Typical products: Control software for washing machines, hi-fis, mobile phones.
Target platform: Small custom-made devices with very limited resources.
Development environment: Since you work with custom made devices, the tool chain is also often custom made. Frequently it's not very advanced at all, compared to the relative luxury of the applications programmer. (As the market broadens we are seeing improvements here.) The code is developed in a *cross-compilation* environment, where target platform is different from host compilation environment. (Clearly you can't compile C on a washing machine. Yet.)
We write specialised software for each specific device. Almost universally embedded programming uses C, apart from really low level work which tends to use assembler. C++ is making inroads into this area, and ADA has also been used.
Common problems/challenges: There are all sorts of problems you can encounter, largely depending on whether you are working with a commodity "off the shelf" embedded platform or building your own. There are issues of real-time programming (for example, timely handling of hardware events and interrupts), direct hardware interfacing, and controlling peripheral connections, plus tedious low-level concerns like byte endianness and physical memory layout.

Distributed programming

Distributed programming develops systems that comprise of more than just the one computer. The World Wide Web is effectively a huge distributed system with information being stored on many computers across the world, and applications being available remotely via your web browser (for example, eBay and Hotmail). It's not all about web browsers, though. Working with and designing distributed systems brings in whole world of new problems.

You might need to 'distribute' a system for a number of reasons. Perhaps some types of computer are more suited to particular tasks than others. Perhaps the system is in high demand, and you can share the workload out among many machines on a network to improve performance. Perhaps there are physical location restrictions for certain machines that mandate you must distribute the system.

Effectively, you design a system that comprises a number of programs on different machines that all work together as a cohesive whole. The RC5 and SETI screensaver projects are excellent examples of this kind of work. Each machine in the system is designated a particular task (or tasks), each task is developed for its target machine, and then brought together in the final system.

These disparate parts need to be glued together somehow; each of the programs need to be able to effectively call functions on remote machines as if they were locally linked to their code. This is known as *remote procedure call* (RPC), and such facilities are provided by a number of available *middleware* technologies. These act as brokers for data transfer between machines, they describe how you discover and talk to services on other machines and how you publish your services for other programs to call. There are security issues (whose allowed to call who?), network latency issues (what happens if a remote function call takes too long?), considerations for balancing synchronous remote function calls with asynchronous calls, and more.

Some of these middleware systems employ object-oriented technologies, some take more of a procedural approach. The middleware is simply "connectivity software" and allows some degree of platform neutrality. As long as the middleware runs on a given platform, the calling code shouldn't care what that platform is – it could be a ZX spectrum for all we care – you will call functions on it in the same way. Of course, in the design of a distributed system you will select the appropriate hardware for each task. It's doubtful you'll see any ZX spectrums knocking about!

Commonly used middlewares are CORBA, the Java RMI and Microsoft's DCOM. Using these we split the system between user interface elements, the 'business logic' (real workhorse code), and any storage required (e.g. a database and query engine). These days the user interface will often involve a web front end.

Typical products: Online purchase system, splitting work between front-end applications (web interface, in-shop kiosk, and/or phone ordering system), business logic (manages stock control, implements ordering system, and separate secure billing) and the shared storage.
Target platform: Many different computer systems connected via a middleware, almost always sitting on top of standard networking protocols.
Development environment: Many and varied. This will depend on languages used, the nature of each computer in the system and the type of middleware employed. Often remotely callable interfaces are defined in some from of *interface definition language* (IDL) and 'compiled' to an implementation language representation that provides all the calling glue and provides hooks for each function implementation to be slotted in to.
Common problems/challenges: Correct split of services between computers, and of the communications involved. This can severely affect the "scalability" of a distributed system. What works for a few transactions a day may not work efficiently for 100 transactions a minute. This sees a real need to design carefully. You also have to deal with computer availability and deal gracefully if one of the computers in the system becomes unavailable.

Enterprise programming

Enterprise is one of those cheesy buzzwords floating around at the moment, more management speak than any programmer dialect. An enterprise is literally a business organisation. So enterprise programming is providing systems for entire companies, gluing all their separate systems together to form a unified cohesive whole. Enterprise programming almost always means the development of large distributed systems.

They'll commonly be deployed on a company intranet (internal network), and join the different 'departments' of the business together to improve workflow. The systems may or may not be customer facing. Once the organisation is running a computer system it's generally not too hard to have automated customer interaction. Perhaps an enterprise system will need to interface to other companies' systems too, for example to track the delivery status of goods.

Typical products: System for an entire company, managing its normal business operations.
Target platform: A tailored distributed system.
Development environment: As for distributed systems. We'll probably be working with huge data stores, perhaps various database technologies from previous internal systems ('legacy systems' in manager speak). XML is all the rage here.
Common problems/challenges: As for distributed systems.

Numerical programming

This kind of work involves scientific, highly technical tasks making heavy use of mathematics. This is another highly specialised area, which requires writing applications specifically targeted at particular numerical problems. The programs are often aimed at supercomputers, the fastest type of computer, capable of massive number crunching operations. Although we're still living in times where the 'fastest' type of computer is changing from year to year, these are very expensive platforms, employed for specialized applications that require immense amounts of mathematical calculations.

[concluded at foot of next page]

My First Steps in Java Web Development

By Emma Willis

Background

As a new member to the ACCU, my software development experience is limited to Java although I am hoping to learn a great deal about C and C++. I started programming in October 2001 for my postgraduate degree, where I was immediately introduced to Java as opposed to C or C++. All of my university projects, especially web-based activities, have been strongly geared towards Java technologies. Consequently, I hope that this article will serve as an introduction to Java Server Pages (JSPs) for many of you, demonstrating how easily and quickly they can be picked up from scratch.

The Project

The project, which is currently in the testing stage, is an evaluation version of a web-based flight and hotel booking system to be used by a new (but sadly imaginary!) low-cost airline — TinyJet. The project requirements determined our use of Java technologies to implement a three-tier website.

As with most university projects, the objective was to test our project management and team-building skills as opposed to the precision of our code; however, for me, I saw this project as the biggest challenge I have been faced with on the course, and I was damned if I was going to let the whole team-building thing stand in the way of us producing the best website imaginable!!!!

By that I mean that I have given a great deal of time to properly researching JSP web development in order that I could pass on the knowledge to my team members. I would strongly recommend Wrox's "Beginning JSP Web Development" (1) for anyone who is taking their first steps with JSPs. Wrox have a reputation for making things more complicated than necessary, but I found this book to be relatively gentle, aiding me with incredibly useful examples. By the end of the 9-week development stage, I would confidently state that all 6 team members were able to produce interactive and dynamically generated web pages using JSPs.

JSPs vs Servlets

So why did we choose to develop the TinyJet system using JSPs as opposed to Java Servlets? There are two schools of thought when it comes to this decision. JSP technology is an extension of the Servlet technology: as such, when a JSP is loaded into the browser, it is compiled within your Servlet engine and converted into a Java Servlet. Members of the first party believe that Servlets are easier to code because the developer has to think their way through the program using the code that the page is produced in. These people see JSPs as more awkward to code because it introduces another style of Java coding on top of the more traditional Java used in Servlets. Members of the second party believe that JSPs are easier for non-Java programmers for precisely the same reason!!! These people believe that the further level of abstraction away from Java coding enables web-page designers with little development experience to work on the design without having to worry about the program logic in the lower tiers.

One of the easiest ways to distinguish between the two development styles is this — writing Servlets is like writing a Java program but writing JSPs is like writing an HTML webpage.

A basic Servlet must import the `javax.servlet` libraries used for developing http applications. It must also extend the `HttpServlet` class provided in the library in order that you may override the `doGet` and `doPost` methods of the webpage.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class EmmasServlet extends HttpServlet
{
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
        throws ServletException, IOException {
    //dynamically prints html
    //java methods can go here
    out.println("<html><body>");
    out.println("Emma's first servlet");
    out.println("</body></html>");
    out.close();
  }
}
```

In contrast, a JSP is written using tags, similar to those used in html. In fact, a JSP need not ever refer to Java or JSP code at all, there may be times when you choose to represent your simple html page using a .jsp file ending, in which case the tags WILL BE html tags. In general, html and JSP code can be mixed together, placing the JSP code within script tags. There is no need to define the `doGet` and `doPost` methods or to place methods within `try` and `catch` blocks to deal with exceptions.

```
<%@ page import='java.io.* %>
<%@ page import='java.util.* %>
<html>
<body>
  Emma's first JSP.
</body>
</html>
```

Professionalism in Programming (continued from page 14)

Weather forecasting, for example, requires a supercomputer (or perhaps a gift of prophesy!). We also see supercomputers used for animated graphics, fluid dynamic calculations, and other areas that require highly complex mathematical investigation and calculation. A supercomputer is not a mainframe. The latter is a high performance computer designed to execute as many programs as possible concurrently, often used as a centralised computing resource in a business setting. A supercomputer channels all its power into executing a few programs as fast as possible. There are a number of different supercomputer architectures exploiting different technological advances, each requiring different algorithmic approaches to fully exploit their power.

This kind of work requires high performance algorithms to get as many calculations done as possible, to really capitalise on the performance of the computing platform. It is common to make use of carefully designed, heavily optimised numerical libraries, and to make explicit use of parallel processing, designing this into the computational algorithms and processes. This will involve both task and data parallelism, either performing many similar tasks on many CPUs at once, or pipelining the algorithm, performing different parts of it on different CPUs.

Typical products: Fields involving highly complex mathematical investigation, for example nuclear energy research or petroleum exploration.
Target platform: Often supercomputers.
Development environment: Although there is work on advancing numerical programming support in C++, and some of this kind of work is performed in C, a lot of numerical programming is done in Fortran.
Common problems/challenges: Crafting efficient algorithms to really exploit the power of the supercomputer.

Conclusion

Of course there are other areas than these, some pretty well defined, others more ephemeral. I can think of *safety critical systems* as one example.

There is a commonality we can observe: each of these different development realms requires fundamental design decisions to be made to suit software to them. Applications level code is not generally suited to an embedded environment, or a workstation application design may not scale when applied to a distributed system. This means that software developers tend to specialise in particular fields, and learn to think in particular patterns that suit their 'world'. Understanding the very real concerns of each environment will make you a more flexible and mature programmer.

Pete Goodliffe

For TinyJet, the decision between Servlets and JSPs was made for us. The team already had limited experience with Servlets and so it was clear that the project was supposed to test our abilities with the newer technology. This caused problems for some members who were terrified at the thought of moving from one technology to another — however, once we had started, the shift from Servlets to JSPs proved an easy transition.

In order to implement JSPs, or indeed, Java Servlets, you will need to run a Servlet engine. We used Jakarta's Tomcat although J-Run is also a common choice for small or personal projects. Tomcat can be downloaded for free from `http://jakarta.apache.org/tomcat`. The Servlet engine can function as a web-server or can be run alongside your existing web-server. I should perhaps point out that members on our course experienced difficulties with different versions of tomcat working alongside different versions of the Java SDK. One combination that definitely works is the latest Tomcat version (about 4.0) and an older version of the Java SDK (1.2.1).

The Tomcat Servlet engine requires a fixed directory structure that should be established before you start; this requires that web-pages (either HTML or JSP) be put in a different folder to Servlets or Java Beans which are held in a 'virtual directory'. It is best to check your Servlet engine documentation.

3-Tier Web Applications - An OO Approach:

The TinyJet project is a three-tier web application. The Java code and web pages sit above two Access databases that store information about Flight times, Flight and Hotel availability, Customer details and Booking details.

Our challenge was to develop, using JSPs, an application that used a number of Java classes to read from and write to these databases. As such, we have tried as far as possible to keep the JSPs responsible only for formatting information on the web page. We have used a central tier of Java 'Beans' or specially written stand-alone Java classes which actually perform the business and programming logic that enables information to be put into and taken out of the databases.

The way that we approached this development was to write classes that modelled the real-life entities identified in the first paragraph of this section: Flight, Hotel, Customer and Booking. We developed different `Customer` and `Booking` classes for the Hotel part of the system as the Hotel database required different kinds of information to the Flight database. For this reason, we developed abstract classes to represent a `Customer` and a `Booking`, and extended these depending upon whether the information was for a `HotelBooking` or a `FlightBooking`.

In addition to these main classes, we developed two further classes that enabled the developer responsible for producing JSPs to print out lists of the `Hotel` or `Flight` objects. These `FlightList` and `HotelList` classes contained methods that read from the database, creating `Vector`s of `Flight`s or `Hotel`s objects. The JSP, having called the method, was able to use a `for`-loop to run through the `Vector`, one object at a time, printing the required attributes to the webpage. This is a very useful tool for printing out a list of available `Flight`s or a list of `Hotel`s in a given destination.

```
<%
for (int x=0; x< myHotel.cityHotels.size();
     x++) {
  Hotel tempHotel =
        (Hotel)myHotel.cityHotels.get(x); %>
  Name:  <%=tempHotel.getName()%> <BR>
  Price: <%=tempHotel.getPrice()%> <BR>
  <IMG SRC='<%=temp.getPicPath1()%>'
  WIDTH=100> <BR>
    Facilities: <%=tempHotel.getFacilities()%>
                <BR>
  <%
}
%>
```

The most important aspect of developing Java classes or 'beans' for use with JSPs is to remember to provide `get` and `set` methods for all variables. The JSP tag library provides tags not only for instantiating classes but also for setting and getting object properties, given that a `set` or `get` method is defined for them:

```
//instantiates a Hotel
<jsp:useBean id='myHotel'
class='HotelPackage.Hotel' />
//assigns a city to a Hotel
<jsp:setProperty name='myHotel'
property='city' value='Amsterdam' />
//gets a Hotel's name
You selected: <jsp:getProperty name='myHotel'
property='title' /><BR>
```

The TinyJet application makes frequent use of the web user's session object using the `session.setAttribute()` and `session.getAttribute()` Java methods to store the user's selected options. It also makes good use of `java.sql.*` and `java.text.*` methods that enabled us to convert date and currency formats from the Access database into UK Java date and currency formats that could be printed to the web page; something that can take quite a while to get your head around.

Outlook

The TinyJet system was a fast application to develop. Even though we had no prior JSP experience, the actual pages were very simple to construct once the middle-tier classes had been developed. We were able to apply attractive web designs using traditional web tools such as DreamWeaver, and we also found it easy to integrate client-side scripting with the server-side JSPs.

I would suggest that those looking to start web development in Java begin with JSPs as opposed to Servlets. I found the JSPs to work very similarly to html pages and so an understanding of the http and Servlet methods wasn't necessary. For those people with prior Java experience, the transfer of your Java knowledge to JSPs should be easily achieved although a background in basic web development would be needed. When it comes from moving from C or C++ to Java web development tools, I am afraid I am no help! I do believe however that if I can pick it up as a beginner in programming, it can't be too daunting for those with previous programming knowledge (I hope!) My advice is not to be worried about the transition because in my experience from TinyJet, it's the worry that holds people back, not their ability.

Emma Willis

References

(1) Falkner. J., et al. (2001). *Beginning JSP Web Development.* Birmingham: Wrox Press.

Don't Touch That Clock!

by Silas S Brown <ssb22@cam.ac.uk>

A friend wanted to transfer a large amount of data from her account in America, so I scheduled a download to take place overnight, using `'wget'`, a good mirroring tool. In the morning, I found this:

```
calc_rate: Assertion `msecs >= 0' failed.
```

Apparently, it was trying to calculate the transfer rate, and found it to be negative. This happened at 2:30am, which is the time when my system synchronises its clock to the university's NTP time server. I checked the logs and it seems the NTP synchronisation took the system clock back 6 seconds, and this must have caused the negative time difference.

I looked through the `wget` source and found it calls the Unix function '`gettimeofday`' to calculate the time to the desired accuracy. Clearly it would be better to use a timer that is relative and not affected by clock changes, but there doesn't appear to be such a thing in the GNU C library.

Apparently, any programmer who deals with elapsed times on a multi-tasking system has to be aware of the fact that somebody might change the system clock while the program is running. This doesn't strike me as an ideal situation.

Of course, it is possible to work around the problem by ignoring unusual timings, or disabling the progress indicator altogether (since it's not really needed in a batch job), or disabling the NTP program that night, or telling it to "slew" the clock instead of setting it (this means that the clock's speed is slightly changed for a while until it has gained or lost the required amount of time, so that the clock never "jumps", but it can take a long time to perform). Still, it seems odd that a system such as Unix should not differentiate between absolute clock times, which might be re-synchronised, and relative timers which should not be affected. Most PDAs manage fine no matter how often you set the clock.

Silas S Brown

Examining C++[1]

by Alistair Merideth <alisdairm@btinternet.com>

Recently I have been involved in both sides of the recruitment process, and noticed there are a couple of popular questions at an interview for a C++ position that are pretty similar wherever you go. These questions are not designed to pick out the star candidates, but rather identify someone whose C++ knowledge may not be all this is desired. Unfortunately, by exploring a little beyond these simple questions I discovered that frequently the interviewers themselves don't really understand the topics they are examining. In this short series of articles we will examine this difference between learning and understanding by looking the classic form of two interview questions and exploring some alternative solutions. We begin with the most frequently encountered question with the most frequently misapplied solution:

Do you see any errors in the following code?

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

class base {
public:
  base( int nData = 0 )
    : m_pnData1( new int( nData ) ) {}
  ~base( void ) { delete m_pnData1; }
  virtual int Data( void ) {
    return *m_pnData1;
  }
private:
  int *m_pnData1;
};

class derived : public base {
public:
  derived( int nData = 0 )
    : base( 5 )
    , m_pnData2( new int(nData) ) { }
  ~derived( void ) { delete m_pnData2; }
  int Data( void ) { return *m_pnData2; }
private:
  int *m_pnData2;
};

int main( void ) {
  base *pBase = new derived(12);
  cout << "Data = " << pBase->Data() << endl;
  delete pBase;
  return 0;
}
```

The trick we are looking for is that base does not have a virtual destructor. Consequently, when `pBase` is deleted it does not call the destructor of `derived` and there will be a memory leak. The solution is therefore to fix base.

```
  virtual ~base( void ) {
    delete m_pnData1;
  }
```

This is as far as a 'standard' interview will take this question. However, what happens if instead we rewrite `main` as follows?

```
int main( void ) {
  derived example(12);
  cout << "Data = " << example.Data() << endl;
  return 0;
}
```

Now we have a simple instance of derived. Do we still have a problem? Will `m_pnData1` in the base class truly be deleted, or will something even more bizarre happen?

The answer is of course that there is no problem with this code now. It will work as we intended and there is no leak. Frequently though, developers I meet are so taken with the virtual destructor mantra that they will insist the only way to fix this problem is to make all destructors virtual. Often they cannot see that the second solution will indeed run correctly. Typically candidates who have not seen the solution phrased this way before are uneasy, perhaps coming to accept the answer but with a feeling that they are being fooled and that really 'something else' will happen although they can't see what. I put this down to seeing and learning the same example over and over without any attempt being made to really teach an understanding of what is happening.

Now if they have really been paying attention a star candidate will complain at this point that although it solves the problem at hand, this code is still dangerous. `derived` is still dangerous to use through a pointer to base as seen in the original problem, and we have not stopped people using it in this manner. This is a good point, as a fragile solution that relies on people strictly following coding conventions is simply storing up a problem as a project evolves. One day someone new will join the team, not knowing all the conventions, and make this mistake. You then have a very hard bug to track.

However, there is one further change we can make that will allow us to keep the second solution intact. The problem is that although it is fine to delete a pointer to `derived`, it is dangerous to call `delete` on a pointer to `base`. To enforce correct usage, we make the destructor of `base` protected. Now the compiler will catch any attempt to delete a `base` for us, and the compiler is an unforgiving observer of coding conventions when we let it know about them!

Some of you may still be wondering if all we have done here is move the problem from `base` to `derived`. This is another good observation, and in the simple example provided the best solution is indeed to make the `base` destructor virtual. However, it is important to understand the second solution as it is a valid idiom and you may well meet it in others' code. More importantly, by understanding it you become aware of when to use it yourself.

So when might we want to use this protected non-virtual destructor idiom? After all, if virtual destructors are so important why aren't they forced on us by the language? Are there any well-known examples of classes without virtual destructors? A quick look into the Standard Template Library will reveal that many of these classes do not have virtual destructors. However, contrary to the advice in this article their destructors are also declared as public. Surely this can't be some mistake in the standard?! Indeed it isn't, and understanding why will lead us to a possible use of our idiom.

The classes in the STL are 'value' classes, where the classes are intended to be used as simple values, 'just like the ints' [4]. These classes need to be efficient with minimal overhead if they are to be used in this manner, otherwise we may look for hacks such as passing pointers to the values around when efficiency really matters to us. A virtual destructor brings no advantage to such a class, as they are not intended to be derived from (any more than you might derive from an `int`, if such a thing were possible [2]. However it does bring overhead in the form of a virtual function table, and all calls to destroy these objects now go by virtual dispatch involving an indirect call through a function pointer that also kills the opportunity for inlining. This may not sound like much, but when you have thousands of these objects with a short lifetime it quickly adds up. So by design a value-class has a no virtual functions and a public, non-virtual destructor [4], and is not intended for use as a base class. Clearly, this does not match our intended use of `base` as a base class, so although this is the best-known example for non-virtual destructors it is not the answer we are looking for.

Looking the other way, typical use of a public virtual destructor is when a base class provides a polymorphic interface for a family of classes. The standard example here is the vector drawing package with the classes circle, square and triangle all deriving from an abstract interface shape. The program works with references to shapes, which it typically obtains through calls to a class factory [5][6] of some kind. The concrete objects are always owned and manipulated through these references (typically some kind of smart pointer). The base class provides the interface, and a required part of that interface is a public destructor in order to manage the lifetime of the objects. So clearly our idiom will not apply here either, as we cannot expose the public destructor.

Where does this leave us? We have a class intended for use as a base class, but without public destructors. The only place we can really use this is as an implementation detail of another class. The subclass derives privately from the base, as it is an implementation detail and nothing to do with the public interface. A good example might be an application of the Template Method pattern [7] (nothing to do with C++ template syntax, despite the name!) For example let's design a report-printing class. We know that when we print a report we will want to print a title page, the content of the report, and then the bibliography. This can be encapsulated into a (non-virtual) `Print` method that calls three abstract virtual functions, `PrepareTitlePage`, `PrepareReport` and `PrepareBibliography`. In order to implement a report now all we need to do is derive privately from the`ReportTemplate` class and override these three functions. The routine details of taking the result of these operations and getting them to the printer are handled by the base class `Print` implementation itself. Clients of our reports don't care that they

My Recommended Books Part 2

by Paul Grenyer <pjgrenyer@iee.org>
(with more than a little help from a lot of friends!)

Before we continue, here is the bit that got squeezed out last issue.

C++ Books continued

C++ FAQs (Second Edition) Marshall Cline, Greg Lomow, Mike Girou
Paperback - 602 pages 2nd Ed (23 February, 1999)
Addison Wesley; ISBN: 0201309831 Price: £30.99

From the back cover...

In a concise and direct question-and-answer format, C++ FAQs, Second Edition brings you the most efficient solutions to more than four hundred of the practical programming challenges you face every day. Moderators of the on-line C++ FAQ at `comp.lang.c++`, Marshall Cline, Greg Lomow, and Mike Girou are familiar with C++ programmers most pressing concerns.

In this book, the authors concentrate on those issues most critical to the professional programmer's work, and they present more explanatory material and examples than is possible on-line. This book focuses on the effective use of C++, helping programmers avoid combining seemingly legal C++ constructs in incompatible ways. This second edition is completely up-to-date with the final ANSI/ISO C++ Standard. It covers some of the smaller syntax changes, such as "mutable"; more significant changes, such as RTTI and namespaces; and such major innovations as the C++ Standard Library, including the STL.

In addition, this book discusses technologies such as Java, CORBA, COM/COM+, and ActiveX – and the relationship all of these have with C++. These new features and technologies are iconed to help you quickly find what is new and different in this edition. Each question-and-answer section contains an overview of the problem and solution, fuller explanations of concepts, directions for proper use of language features, guidelines for best practices and practices to avoid, and plenty of working, stand-alone examples. This edition is thoroughly cross-referenced and indexed for quick access.

Why this book made it into the list...

It's so useful for intermediate level programmers that in my last job I persuaded the company to buy copies for each of their 20-odd developers. Again, it's broken into many small chunks, which means that people can dip into it during their working day without having to keep track of the big picture. — James Dennett, `accu-general`

Windows/COM/ATL Books

Programming Applications for Microsoft Windows Jeffrey Richter
Hardcover - 1056 pages Bk&Cd Rom (September 1999)
Microsoft Press; ISBN: 1572319968 Price: £56.49

From the back cover...

Build Powerful Win32-based applications and prepare for you 64-bit future.

Here's the definitive instruction for advancing the next generation of windows based applications - faster, sleeker and more potent than ever. This fully updated version of the best selling Advanced Windows digs even deeper into the advanced features and state of the art techniques you can exploit for more robust Windows development - inlcuind authoritative insights on the Windows 2000 operating system.

Coverage Includes:

- Understanding Windows 2000 features - such as jobs, thread-pooling PIs, Addressing Windowing extensions, Toolhelp functions and sparse files.
- Mastering DLL basics and applying advanced techniques - including API hooking, DLL injection, function forwarders, delay loading, redirection, rebasing and binding.
- Solving common thread-synchronisation problems with a toolkit of package reusable code.
- Building high performance scaleable applications by understanding data alignment, cahce-line boundaries, cross-process critical sections, NUMA architectures and processor infinity.
- Using structured exception handling to create memory efficient applications.
- Transitioning to 64-bit development - see what's ahead by studying a full cache of code built and tested on Windows 2000.

Why this book made it into the list...

Extensive, yet engaging, no holds barred coverage of the lower level Windows API services that go much beyond the SDK specifications. Essential reading and reference for the Windows system programmer. — Phil Nash, `accu-general`

Inside COM Dale Rogerson
Paperback - 350 pages (1996)
Microsoft Press International; ISBN: 1572313498 Price: £25.99

From the back cover...

COM without the complexity.

Microsoft's Component Object Model (COM) has emerged as a vital tool. It's the basis of Microsoft's Approach to distributed computing. It's a powerful method of customizing applications, present and future. And it's the foundation of OLE and ActiveX. In short COM helps unlock the future or development. And the is the book that unlocks COM. In it, you'll discover:

- A clear and simple, practical guide to building elegant COM components.
- An eye-opening presentation of how accessible COM can be - especially for those who have already mastered C++.
- An insightful, progressive view of COM design
- Plenty of examples in the form of code samples

Inside COM is for intermediate to advanced C++ programmers; COM; ActiveX and OLE programmers; academics with an interest in computer design; and programmers who want to use COM when it is ported to UNIX, MVS and other environments. To put it simple, COM-based interfaces are spreading fast - and if you work with any of them, inside COM is written for you.

`[continued from previous page]`

are implemented in terms of a `ReportTemplate` class, they simply want to create an instance of our `AccountsReportPrint` class and run the print job.

In summary the destructor for a value class should be non-virtual and public, whereas the destructor for a base class should be either virtual and public, or more rarely non-virtual and protected. It is left as an exercise to the reader to find a use for private destructors!

Hopefully this article has shown you that preparing good interview questions is hard, and dealing with the answers can be harder! Along the way you have picked up or confirmed your ideas on the design decisions to consider when deciding how to declare the destructor for your classes, and why simply declaring everything virtual is not the only way.

The conscientious among you are already worrying about a host of other fragilities in the example classes. Have no fear! These will be addressed in the next two articles on Temporary Values and Exception Safety.

Alistair Merideth

Footnotes

[1] The title is an affectionate nod to two classic books, Effective C++ and Exceptional C++, where I first learned most of the ideas contained in these articles, along with the value of a good book.

[2] Actually this IS possible in other languages where everything is an object, such as Java or Smalltalk, although I am not aware of this being a popular thing to do! (and may be one motivation for the 'final' keyword in Java)

[3] Microsoft C# enshrines this difference between value types and reference types directly in the language.

[4] Effective C++, Scott Meyers. Published Addison Wesley 1998? (second edition)

[5] Factory pattern, Design Patterns by Gamma, Helm, Johnson and Vlissides. Published Addison Wesley 1995

[6] Factory pattern, Modern C++ Design Alexei Alexandrescu. Published Addison Wesley 2001

[7] Template method, Design Patterns by Gamma, Helm, Johnson and Vlissides. Published Addison Wesley 1995

Why this book made it into the list...

I was presented with this book in my first week of my first job, when I had never even heard of COM and told it would tell me everything I needed to know. How right they were. I find this book very easy to read and understand. It doesn't deal with any sort of boiler plate 'wizard' and teaches the reader exactly what is happening within COM and how to derive and implement COM objects from the very basics. — Paul Grenyer

Essential COM **Don Box**
Paperback - 256 pages (2 February, 1998)
Addison Wesley; ISBN: 0201634465 **Price: £30.99**

From the back cover...

"Don Box makes it possible for mere mortals to join the COM cognoscenti. If you're a C++ COM programmer, buy this book." -David Chappell, Principal, Chappell & Associates and author of Understanding ActiveX and OLE.

Written by a leading COM authority, this unique book reveals the essence of COM, helping developers to truly understand the why, not just the how, of COM. Understanding the motivation for the design of COM and its distributed aspects is critical for developers who wish to go beyond simplistic applications of COM and become truly effective COM programmers. As the COM programming model continues to evolve, such insight also becomes essential to remaining current with extensions, such as Microsoft Transaction Server and COM+. By showing you why Distributed COM works as it does, Don Box enables you to apply the model creatively and effectively to everyday programming problems.

This book examines COM from the perspective of a C++ developer, offering a familiar frame of reference to ease you into the topic. You will also find comprehensive coverage of the core concepts of Distributed COM (interfaces, classes, apartments, and applications), including detailed descriptions of COM theory, the C++ language mapping, COM IDL (Interface Definition Language), the remoting architecture, IUnknown, monikers, threads, marshalers, security, and more. In addition, the book offers a thorough explanation of COM is basic vocabulary, provides a complete Distributed COM application to illustrate programming techniques, and includes the authors tested library of COM utility code.

Why this book made it into the list...

The reason this book made it into the list is very, very simple. Every time I posted a COM question to `accu-general` (which was quite frequently) the author of the reply would refer to Essential COM, by Don Box. — Paul Grenyer

Effective COM **Don Box, Keith Brown, Tim Ewald, Chris Sells**
Paperback - 224 pages Reissue (30 June, 1999)
Addison Wesley; ISBN: 0201379686 **Price: £34.99**

From the back cover...

In Effective COM, the authors, Don Box, Keith Brown, Tim Ewald, and Chris Sells, offer 50 concrete guidelines for creating COM based applications that are more efficient, robust, and maintainable.

Drawn from the authors extensive practical experience working with and teaching COM, these rules of thumb, pitfalls to avoid, and experience-based pointers will enable you to become a more productive and successful COM programmer. These guidelines appear under six major headings- the transition from C++ to COM; interfaces, the fundamental element of COM development; implementation issues; the unique concept of apartments; security; and transactions.

Throughout this book, the issues unique to the MTS programming model are addressed in detail. Developers will benefit from such insight and wisdom as:

- Define your interfaces before you define your classes (and do it in IDL).
- Design with distribution in mind Dual interfaces are a hack.
- Don't require people to implement them.
- Don't access raw interface pointers across apartment boundaries.
- Avoid creating threads from an in-process server.
- Smart Interface Pointers add at least as much complexity as they remove.
- CoInitializeSecurity is your friend. Learn it, love it, call it .
- Use fine-grained authentication .
- Beware exposing object references from the middle of a transaction hierarchy.
- Don't rely on JIT activation for scalability.

...and much more invaluable advice. For each guideline, the authors present a succinct summary of the challenge at hand, extensive discussion of their rationale for the advice, and many compilable code examples. Readers will gain a deeper understanding of COM concepts, capabilities, and drawbacks, and the know-how to employ COM effectively for high quality distributed application development.

Why this book made it into the list...

...Written in the point by point short advice then explanation style of Scott Meyers' books. A slim volume but packed with useful information about more than just DCOM. — Garry Lancaster, `accu-general`

If you are looking at doing a lot more with COM/ATL then I would suggest two books for you. 1. Effective COM, Don Box et al. 2. ATL Internals, Rector, Sells. — David Williams, `accu-general`

Learning DCOM **Thuan L. Thai**
Paperback - 504 pages (April 1999)
O'Reilly UK; ISBN: 1565925815 **Price: £23.50**

From the back cover...

This book introduces C++ programmers to DCOM and gives them the basic tools they need to write secure, maintainable programs. It clearly describes the C++ code needed to create distributed components and the communications exchanged between systems and objects, providing background, a guide to Visual C++ development tools and wizards, and insight for performance tuning, debugging, and understanding what the system is doing with your code.

Why this book made it into the list...

There are a lot of good COM and ATL books out there, but all of the ones I've read so far only really scratch the surface of DCOM. This is another COM book like any other in as much as it discusses the fundamentals of COM and how to use boiler plate tools such as the ATL. Where it really scores above the other books is its section on client techniques. — Paul Grenyer

ATL Internals **Brent E. Rector**
Paperback - 544 pages Reissue (21 April, 1999)
Addison Wesley; ISBN: 0201695898 **Price: £41.99**

From the back cover...

"Brent and Chris are always technically accurate and present the information in a well written, easy to understand manner... I bought it and I'm the ATL Development Lead!"—Christian Beaumont, ATL Development Lead, Microsoft Corporation.

The Active Template Library (ATL) is a set of small, efficient, and flexible classes that facilitate the creation of interoperable COM components. Written for experienced COM and Visual C++ programmers, this book provides in-depth coverage of ATLs inner workings. It offers insight into the rationale behind ATL design, explains its architectural underpinnings, shows how ATL maps to COM, and describes important implementation details.

With coverage current through ATL version 3.0, ATL Internals includes an overview of the Wizards but then goes well beyond the basics. The authors provide the detailed information needed to utilize ATL to its greatest advantage and work around its shortcomings. You will find detailed coverage of such topics as:-

- ATL Smart Types, such as CComPtr, CComVariant, and CComBSTR
- Objects in ATL, covering COM object responsibilities and ATL threading model support .
- Servers in ATL, including ATLs class object implementations, managing server lifetime, self-registration, and server build optimisations.
- Interface maps, focusing on the techniques C++ programmers can use to implement COM interfaces and how ATL supports these techniques.

- Persistence and connection points Enumeration, covering both pre-calculated and dynamic data sets, and enumerating over an STL container ATL windowing classes.
- Controls and Control Containment.

If you want to optimise ATL by learning effective techniques that reduce the time you spend writing boilerplate COM code, there is no better resource than this book.

Why this book made it into the list...

It would worry me if I could not recommend this book, the foreword is written by the inventor of ATL, while the sales blurb on the back reads like a 'who is who' of ATL and COM. I was not surprised to find that it lived up to my expectations. The book is very much in depth, in places I have often wondered if one really needs to know ALL the detail. I have never yet needed to know some of the things that this book covers and I do not expect I will, however if you often have to work with multibyte character sets you will appreciate the in-depth discussion of the appropriate types. For technical in depth detail of ATL this may be your book, but it is certainly not ideal for learning ATL. — John Crickett, ACCU Book Reviews

COM IDL and Interface Design **Al Major**
437 pages (February, 1999)
Wrox Press; ISBN: 1861002254 **Price: £45.99**

From the back cover...

This book clearly explains the syntax and usage of IDL, but that's only the beginning of the story. You'll also learn how to write efficient interfaces in a way that facilitates their use from languages other than C++. You'll get a comprehensive (over 40) list of interface and object design techniques and guidelines that shorten your design learning curve and pay for the price of the book.

The book places all of this in context by demonstrating C++/ATL code that implements an On-Line Auction. You'll see sophisticated COM techniques, such as Alternate Identity, Delayed Initialisation, Split Identity, multiple scriptable IDispatch interfaces, persistence delegation, marshalling structures with embedded pointers, using IMallocSpy, etc.

Why this book made it into the list...

The contents of the book cover COM and IDL basics, Remote Method Calls, Automation and Tool Support, a particularly good section on Application Design - covering both client and server issues in multiple languages (after all COM components are supposed to be language agnostic) & common COM patterns and protocols, and a case study (in this case an online auction system) tying this all together. Understanding IDL and the benefits of writing it yourself, rather than leaving it to the wizards, is a skill I think all serious (by that I mean day-in day-out users of, like me) COM developers should acquire. This book should be a hyperlink from point 1 in Effective COM. 'Define your interfaces before your classes (and do it in IDL)'. — David Williams, `accu-general`

Programming/Patterns/UML Books

Design Patterns
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
(Also known as Gang of Four or GOF)
Hardcover - 224 pages Reissue (14 December, 1994)
Addison Wesley; ISBN: 0201633612 **Price: £37.99**

From the back cover...

The authors begin by describing what patterns are and how they can help you design object-oriented software. They then go on to systematically name, explain, evaluate, and catalogue recurring designs in object-oriented systems. With Design Patterns as your guide, you will learn how these important patterns fit into the software development process, and how you can leverage them to solve your own design problems most efficiently.

Each pattern describes the circumstances in which it is applicable, when it can be applied in view of other design constraints, and the consequences and trade-offs of using the pattern within a larger design. All patterns are compiled from real systems and are based on real-world examples. Each pattern also includes code that demonstrates how it may be implemented in object-oriented programming languages like C++ or Smalltalk.

Why this book made it into the list...

Everyone I spoke to on ACCU General recommended having at least one patterns book and that this was the one to have. — Paul Grenyer

You probably don't need me to tell you, but this is a very rewarding book. Buy it and dip into it a few times and you'll find yourself coming back to it time and time again. The reward comes from the "lightbulb effect" as patterns start to suggest themselves when you're designing systems later on. — Sean Corfield, ACCU Book Reviews

The Practice of Programming **Brian W. Kernighan, Rob Pike**
Paperback - 256 pages (23 February, 1999)
Addison Wesley; ISBN: 020161586X **Price: £22.99**

From the back cover...

With the same insight and authority that made their book The Unix Programming Environment a classic, Brian Kernighan and Rob Pike have written The Practice of Programming to help make individual programmers more effective and productive.

The practice of programming is more than just writing code. Programmers must also assess tradeoffs, choose among design alternatives, debug and test, improve performance, and maintain software written by themselves and others. At the same time, they must be concerned with issues like compatibility, robustness, and reliability, while meeting specifications.

The Practice of Programming covers all these topics, and more. This book is full of practical advice and real-world examples in C, C++, Java, and a variety of special-purpose languages. It includes chapters on:

- Debugging- finding bugs quickly and methodically testing- guaranteeing that software works correctly and reliably performance- making programs faster and more compact portability.
- Ensuring that programs run everywhere without change design.
- Balancing goals and constraints to decide which algorithms and data structures are best interfaces.
- Using abstraction and information hiding to control the interactions between components style.
- Writing code that works well and is a pleasure to read notation.
- Choosing languages and tools that let the machine do more of the work.

Kernighan and Pike have distilled years of experience writing programs, teaching, and working with other programmers to create this book. Anyone who writes software will profit from the principles and guidance in The Practice of Programming.

Why this book made it into the list...

Pick up a copy, choose any chapter and start reading. I think you will then feel motivated to buy yourself a copy, though you might pretend that it was to give to some pestilential colleague whose code was always unreadable, bug-ridden and untested. Whatever language you program in I think you will benefit from reading this book (even if for some rare readers it is just a warm feeling of having been doing the right things all along). — Francis Glassborow, ACCU Book Reviews

Whilst it is not written for students it should be one of the first books students buy and by inference lecturers should get it before the students do! It will stop students (I hope) gaining some of the 'interesting' processes they seem to collect. It will also speed things up as learning from some one else's mistakes leaves you more time to do what you should be doing. If you are not doing embedded C programming (where you should be using the MISRA-C guide) this will be an invaluable book. It is small and the actual text being clear and concise, easy enough to read. At the back is a very useful list of one line 'rules' sub-divided into categories such as style, interfaces testing, etc. It is the sort of book you can dip into time and time again, as the chapters are self-contained. — Chris Hills, ACCU Book Reviews

The Pragmatic Programmer **Andrew Hunt, Davis Thomas**
Paperback - 300 pages (30 November, 1999)
Addison Wesley; ISBN: 020161622X **Price: £26.99**

From the back cover...

Ward Cunningham come straight from the programming trenches, The Pragmatic Programmer cuts through the increasing specialization and technicalities of modern software development to examine the core process - taking a requirement and producing working, maintainable code that delights its users.

It covers topics ranging from personal responsibility and career development to architectural techniques for keeping your code flexible and easy to adapt and reuse. Read this book, and you'll learn how to:

- Fight software rot.
- Avoid the trap of duplicating knowledge.
- Write flexible, dynamic, and adaptable code.
- Avoid programming by coincidence.
- Bullet-proof your code with contracts, assertions, and exceptions.
- Capture real requirements.
- Test ruthlessly and effectively.
- Delight your users
- Build teams of pragmatic programmers
- Make your developments more precise with automation.

Written as a series of self-contained sections and filled with entertaining anecdotes, thoughtful examples, and interesting analogies, The Pragmatic Programmer illustrates the best practices and major pitfalls of many different aspects of software development. Whether you're a new coder, an experienced programmer, or a manager responsible for software projects, use these lessons daily, and you'll quickly see improvements in personal productivity, accuracy, and job satisfaction. You'll learn skills and develop habits and attitudes that form the foundation for long-term success in your career. You'll become a Pragmatic Programmer.

Why this book made it into the list...

For me this is one of those 'must read' books. Even though much of the content will be obvious to experienced programmers it is always pleasant to have your opinions confirmed by others and when they are as well articulated as you will find them in this book you really should not begrudge the authors their royalties. Overall this book is well written in highly readable English and is full of common sense coupled with insights that maybe new to many readers. For me this is a great book to start the new century, I think you will agree. — Francis Glassborow, ACCU Book Reviews

Refactoring **Martin Fowler**
Paperback - 320 pages Reissue (31 July, 1999)
Addison Wesley; ISBN: 0201485672 **Price: £30.99**

From the back cover...

As the application of object technology-particularly the Java programming language-has become commonplace, a new problem has emerged to confront the software development community. Significant numbers of poorly designed programs have been created by less-experienced developers, resulting in applications that are inefficient and hard to maintain and extend. Increasingly, software system professionals are discovering just how difficult it is to work with these inherited, "non-optimal" applications.

For several years, expert-level object programmers have employed a growing collection of techniques to improve the structural integrity and performance of such existing software programs. Referred to as "refactoring," these practices have remained in the domain of experts because no attempt has been made to transcribe the lore into a form that all developers could use. . .until now.

In Refactoring - Improving the Design of Existing Code, renowned object technology mentor Martin Fowler breaks new ground, demystifying these master practices and demonstrating how software practitioners can realize the significant benefits of this new process. With proper training a skilled system designer can take a bad design and rework it into well-designed, robust code.

In this book, Martin Fowler shows you where opportunities for refactoring typically can be found, and how to go about reworking a bad design into a good one. Each refactoring step is simple - seemingly too simple to be worth doing. Refactoring may involve moving a field from one class to another, or pulling some code out of a method to turn it into its own method, or even pushing some code up or down a hierarchy. While these individual steps may seem elementary, the cumulative effect of such small changes can radically improve the design.

Refactoring is a proven way to prevent software decay. In addition to discussing the various techniques of refactoring, the author provides a detailed catalogue of more than seventy proven refactorings with helpful pointers that teach you when to apply them; step-by-step instructions for applying each refactoring; and an example illustrating how the refactoring works. The illustrative examples are written in Java, but the ideas are applicable to any object-oriented programming language.

Why this book made it into the list...

It's the last book I read that changed my view of how I work with code, both improving existing code and working on new code. — James Dennett, `accu-general`

It's a relatively easy read too, as it has points similar to the "Effective C++" books. — Terje Slettebø, `accu-general`

UML Distilled **Martin Fowler, Kendall Scott**
Paperback - 224 pages 2nd Ed (30 September, 1999)
Addison Wesley; ISBN: 020165783X **Price: £22.99**

From the back cover...

Now widely adopted as the de facto industry standard and sanctioned by the Object Management Group, the Unified Modelling Language (UML) is a notation all software developers need to know and understand. However, the UML is a big language, and not all of it is equally important. The award-winning first edition of UML Distilled was widely praised for being a concise guide to the core parts of the UML and has proved extremely successful in helping developers get up and running quickly. UML Distilled, Second Edition, maintains the concise format with significantly updated coverage of use cases and activity diagrams, and expanded coverage of collaborations. It also includes a new appendix detailing the changes between UML versions.

Written for those with a basic understanding of object-oriented analysis and design, this book begins with a summary of UMLs history, development, and rationale and then moves into a discussion of how the UML can be integrated into the object-oriented development process. The primary author profiles the various modeling techniques in the UML—such as use cases, class diagrams, and interaction diagrams—and describes the notation and semantics clearly and succinctly. He also outlines useful non-UML techniques such as CRC cards and patterns. These descriptions are made even more relevant with a collection of best practices based on the primary authors experience and a brief Java programming example demonstrating the implementation of a UML-based design. With this tour of the key parts of the UML, readers will be left with a firm foundation upon which to build models and develop further knowledge of the Unified Modelling Language.

Why this book made it into the list...

Some time ago I was enquiring on ACCU General about leaning UML and which books were the best. Nearly everyone recommended this book and I am now about halfway through reading it and find it very easy to read and understand. — Paul Grenyer

If Object-Oriented Analysis and Design methodology is important to you (and anyone involved in writing anything beyond simple applications should consider a degree of formal analysis and design important) you will need to get to grips with UML. So all I can say is, start here. At least you can be certain that the skills you develop will be portable. — Francis Glassborow, ACCU Book Reviews

How To Find These Books.

All of these books are available from Amazon in the UK with the exception of COM IDL and Interface Design by Al Major which is available direct from Wrox Press (`http://www.wrox.com/`).

I personally buy all my books from one supplier who offers at least a 10% discount on every book and often a much greater one. So as not to turn this article into (more of) an advert I won't mention their name. However, I would be pleased to pass the details onto anyone who cares to email me.

Paul Grenyer

A Little String Thing

by Paul Whitehead <paul.whitehead@bcs.org.uk>

It is sometimes useful to be able to tidy up a string before further processing in a software system e.g. at the point where the user has entered the string via the user-interface. From this point on an assumption may be safely made that the string contains no formatting "surprises". Perhaps one of the hardest things to spot with strings is leading or trailing white space. As unlikely as it may seem this can cause systems to fail e.g. a table in a database cannot be found because leading white space in a table name is present. Trying to find such issues can really be like looking for a needle in a haystack.

By removing all non-significant white space (by that I mean leading and trailing white space) at the point where the string has just been entered by a user, such problems should not occur. Performing the operation at the point of entry of data into a system means that no further checks for leading/trailing white space need to be made in other parts of the code and provides for a single point of maintenance.

The project I have in mind uses the C++ standard library string class. Despite criticisms from some quarters that the string class has too many methods (a "Swiss army knife" of a string class, if you will) it certainly doesn't have any methods for removing leading and/or trailing white space. This means we have to code it ourselves. Well, the string class does provide "find" type methods so we could use those to locate specific portions of the string and extract the "real" part of the string. Below is some sample code:

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

namespace {
void rem_space(std::string& str) {
  typedef std::string::size_type pos_t;
  pos_t start = str.find_first_not_of(' ');
  pos_t end = str.find_first_of(' ', start);
  str = str.substr(start, end - start);
}

void show(const std::string& str) {
  std::cout << "Text is: *" << str << '*' <<
std::endl;
}

void test_it(std::string& str) {
  show(str);
  rem_space(str);
  show(str);
  std::cout << std::endl;
}}

int main() {
  using std::string;

  string test1("   abc   ");
  test_it(test1);

  string test2("   abc");
  test_it(test2);

  string test3("abc   ");
  test_it(test3);

  string test4("abc");
  test_it(test4);

  return 0;
}
```

On the face of it, the code seems to do the trick. The method `rem_space` in the anonymous namespace is really the heart of it and I guess the three lines of code speak for themselves (note that the clear names of the methods on the `std::string` class make the code almost self-documenting here). The start and end positions are **not** checked against `std::string::npos` here as such checks are unnecessary in this context (but they should be checked for `start > end` — more later).

Still, there's plenty wrong with the code above! For a start, it only checks for a `' '` as white space. What about tab (`\t`) and newline (`\n`) and all other such characters? A more subtle problem with the code is if you give it a string such as `"   abc abc   "`. What would you expect the result to be? What would you want the result to be? (Unfortunately, the two don't always coincide!)

This can easily be fixed by changing the line

```
pos_t end = str.find_first_of(' ', start);
```

to

```
pos_t end = str.find_last_not_of(' ');
```

The work to fix the check for white space becomes a little more involved. We could use a string of characters to look for (a character class for those of you into regular expressions) e.g.

```
str.find_first_not_of(" \t\n");
```

Sure we've got all our white space characters? A quick look at an ASCII table will provide several more so we're definitely lacking here. Alternatively, we could use an already-provided method called `isspace()` which — guess what — checks to see if the character passed to it is a white space character. It even does more than that — it will also use locale information to interpret what should be classed as white space. I won't go into locales here as it's beyond the scope of this article [1]. We end up with a slight problem here: you can't pass a function or functor to the `std::string "find"` methods. Looks like we need to iterate over the string and provide some means of processing it on a character by character basis. `std::find_if` is one way of iterating over a container and applying an arbitrary "find" predicate to each of the values in the container. Fortunately, the `std::string` behaves like a container class in that it supports iterators — in fact, random access iterators — so we can use the general algorithm methods such as `std::find_if`. So here's a slightly modified version:

```
void rem_space(std::string& str) {
  typedef std::string::iterator str_it;

  str_it str_start =
    std::find_if(str.begin(),
                 str.end(),
                 std::not1(isspace));

  str_it str_end =
    std::find_if(str.rbegin(),
                 str.rend(),
                 std::not1(isspace)).base();

  str = (str_start <= str_end) ?
        std::string(str_start, str_end) : "";
}
```

The additional includes are:

```
#include <locale> // for isspace()
#include <algorithm> // for std::find_if
#include <functional> // for std::not1
```

Before you try it: no, it won't compile, but first let's take a look at what we've got here.

Instead of using `std::string::find_nnn()` we are now using `std::find_if`. This is so that we can apply an arbitrary predicate - in our case, `isspace()`. Note that `isspace()` is one of a set of global convenience functions that use facets and locales under the covers.

We are now using `std::string::iterator` types instead of the `std::string::size_type` as `find_if` works with iterators and the `std::string::find_nnn` methods return you numeric positions rather than integers. In brief, the algorithm looks like this:

1) find the first location in the string where we do not have a space
2) find the last location in the string which is non white space
3) construct a temporary string from the start and end iterators and assign this to the input string

Again, no checks need to be made on whether the iterators returned from `std::find_if` are equivalent to `str.end()` as the code will work fine if the start and end iterators are equivalent to `str.end()`. However, there is a possibility that the start iterator will end up with a value greater than the end iterator: consider the case of a string containing nothing but white space. The forward iterator will go all the way to the end of string and return `str.end()`. The reverse iterator will start at the end and go all the way to the beginning, returning `str.begin()`! The ternary operator is used to cope with this case.

[concluded at foot of next page]

Some Personal Reflections on C++ and Java

by Roger Orr <rogero@howzatt.demon.co.uk>

I worked for a while last year on a project which used both C++ and Java and thought some of my experiences in this project might be of interest to others.

There were three of us on the team - one was a Java enthusiast, the other was happier with C++ than Java and I didn't have a particularly strong brief either way. The design phase was somewhat intriguing – we had little difficulty with the high-level architecture - deciding how to split the activity into a set of processes running on various machines and coming up with a simple inter-process communication method. The part we argued over more was which language to use for implementing each process!

Life might have been simpler if we'd stuck with one language, but there were good reasons for wanting a mix.

Which is the best tool for the job?

Deciding what is the 'best language' is a hard question to answer in the general case! Some of the factors which affect the choice of programming language for a given task are:

- the skills you have on the development team
- the likely skills available for maintenance
- what other components you already have and/or need to use
- licence costs (development environment/runtime)
- customer preferences
- performance requirements
- reliability
- development time

not forgetting the other, less acknowledged, factors which are part of most decisions:

- choice by default — didn't realise there were options
- prejudice
- looks good on the CV
- more 'exciting' technology
- and I'm sure you can think of others...

In some of our cases the choice was obvious — for example we wanted to write some Web server code and Java servlets were a good fit. In other cases we required high performance, and had a gut feeling that C++ gave us more control over this than Java — we certainly had more experience between us in optimising C++ code rather than doing the same for Java code.

However in some cases there was no clear winner, so the person doing the first piece of coding for the process got to choose which language to use!

Coding with two hats on

What is different when you are writing code in two languages? First of all, each of us tended to have a 'preferred' language and found the other language slightly harder to program. For this reason it was a good idea to keep away from the more complicated parts of each language - we didn't use templates in C++ for example.

We also found that by keeping to "common ground" between C++ and Java it was possible to share common algorithms between the languages.

The main difficulties were of course the minor syntactical differences between the languages. The compiler catches most of these, although they are still a bit of a pain, but unfortunately the compilers don't help two particular cases so you just have to try and write (and read) the code with your brain fully engaged.

A Little String Thing (continued from previous page)

So, the `str_start` iterator is set to the first value not containing a white space character, starting from the beginning of the string. `str_end` is set to the last character not containing a white space value, starting from the end of the of string – but what is the `.base()` tacked on to the `str_end` line of code? Well, to start from the end of the string we used the `std::string::reverse_iterators` given by `rbegin()` and `rend()`. The type of iterator returned by `std::find_if` will also be a `reverse_iterator` in this case. We want a forward iterator here so we can easily construct our sub-string object from the start and end iterators using the range constructor form of `std::string`. Calling "`base()`" on a `reverse_iterator` performs the conversion of a `reverse_iterator` into a regular iterator for you [2].

Even so, it still won't compile. The problem is to do with the fact that we are passing a "normal" function into the `find_if` as its predicate. Apart from an efficiency consideration [3], the issue is that the free function `isspace()` is not adapatable. This means you cannot apply function adapters to them — such as `bind1st, bind2nd` etc. What we need to do is provide our own function object (aka "functor") that is adaptable. Here's the code for the functor:

```
struct is_space : public
  std::unary_function<std::string::value_type,
bool> {
  result_type operator()(const argument_type&
val) const {
    //The only global isspace() I have access
to returns int
    // - which isn't conformant but is
programming life!
    return isspace(val) != 0;
  }
};
```

Inheriting from `std::unary_function` doesn't do much other than give the `struct` a few `typedef`s — yet, these are the `typedef`s that make the functor adapatable. I've used a couple of the `typedef`s provided by `std::unary_function` in my own `operator()` such as "`result_type`" and "`argument_type`". These just pick up the types I specified in the template parameters to the base `unary_function` class — if I change the types later on, the rest of the code will follow which means less typing for me! Perhaps more importantly, though, it also means once again a single point of maintenance: I don't need to change the argument types in two — albeit closely physically related — places if I use the `typedef`s provided for me by `unary_function`.

Modifying the code yet again to use my `is_space` functor we end up with this:

```
void rem_space(std::string& str) {
  str_it str_start =
    std::find_if(str.begin(),
                 str.end(),
                 std::not1(is_space()));

  str_it str_end =
    std::find_if(str.rbegin(),
                 str.rend(),
                 std::not1(is_space())).base();

  str = (str_start <= str_end) ?
        std::string(str_start, str_end) : "";
}
```

A full version of the final source code is attached as a zip file which the ACCU are free to publish on their website for download if they so wish.

Of course, if you can do it better or have suggestions to improve this version then write in! I'm sure James would welcome your input. *(Absolutely! — ed)*

Paul Whitehead

Footnotes

[1] If you're interested in locales, Josuttis's "C++ Standard Library" has a good section on the subject, whilst the piece de resistance, at least in my opinion, is Klaus Kreft and Angelika Langer's "C++ I/O Streams and Locales"

[2] More on this in, amongst other works, Scott Meyer's "Effective STL"

[3] A function pointer passed as a predicate is called by de-referencing the function pointer whereas a function object (a.k.a. "functor") can usually be made inline

The first problem was string comparisons. C++ programmers tend to write this:

```
string s;
string t;
if( s == t ) {
  ...
}
```

which is of course perfectly OK in C++, thanks to operator overloading, but then translate it into Java as:

```
String s;
String t;
if( s == t ) {
...
}
```

which is not going to work in general since, as any Java programmer knows, this just checks the `s` and `t` refer to the same actual object and not whether the string values are the same. The Java version for the test would be

```
if( s.equals(t) )
```

To make matters worse, sometimes the equality test will work in Java — the `String` class keeps an internal list of so called canonical string objects. These are used for all string literals and constant strings — and may be also used by the programmer via the `intern()` method of the `String` class. For `String` objects from this list, if `s.equals(t)` is true then so is `s==t`.

The other problem — in the other direction — was with throwing exceptions. A Java programmer writes:

```
throw new MyException( "Error" );
```

together with

```
try
  ...
catch ( MyException ex )
```

and thinks nothing of it. However, if the programmer writes the same code in C++ it will compile without warning and run without apparent error — but not do what is expected when the exception actually gets thrown!

Why not? Well, in the Java world the exception is a reference to a `MyException` object, but in the C++ world it is a pointer to a `MyException` object and so the catch statement won't match the type of the thrown object.

There is a second problem too in the C++ code — the exception is being caught by copy and if the actual exception thrown were from a derived class this extra information would be lost.

It is important to ensure the C++ code looks like:

```
throw MyException( "Error" );
```

together with

```
try
  ...
catch ( MyException & ex )
```

This is a hard problem to find — like nearly all problems in exception handling — because the problem only arises when you have an exception. So you then have two problems - firstly (since the exception is not being handled properly) the program typically just aborts, and secondly you still have to find the cause of the original exception. When the exception is caused by a condition that is hard to reproduce it can waste a lot of debugging time. This was one of the places where Java wins hands down over C++ — firstly because the compiler checks the right sort of object is being thrown and secondly because Java exceptions have built in stack tracing and often just printing the stack trace at the top level of the program was enough to find and fix the problem.

Build problems

I always expected C++ would be hard (all these link time dependencies, incompatible compiler settings, etc) but we had more problems with our Java code.

Java code is compiled in one environment (with one set of class libraries) and then executed in a potentially different environment. Programs typically do not start if class files are missing at runtime — which is bad enough but at least it is a fairly easy problem to fix.

A worse problem occurs when a class file found at runtime does not match the class file used to compile the code. Since Java resolves method calls at runtime, the problem usually only shows itself when a missing or changed method is called — producing a runtime exception and the death of the program.

This caused us problems with JAR files too — the order of JAR files on the class path becomes critical if there are any duplicate class files. We discovered that 3rd party JAR files sometimes include a snapshot of a specific version of some Java interface or class — in particular some of the javax.xml packages seemed to get around a lot...

We found that, having statically linked all our C++ programs, we needed to more care in setting up the runtime environment for Java than for C++.

Integration Testing

We neared the end of our first round of development and reached the point where all the processes could be linked together and the system as whole was ready to test.

I was quite surprised with what we found when we started our testing!

Firstly, memory leaks. Now everyone knows C++ tends to leak ... so did we. Hence we'd avoided using raw pointers in C++ code and used other techniques — standard library collections, auto pointers, reference counted pointers. We were pleased with the result — our C++ programs barely leaked any memory.

Unfortunately we had a leak in one of our Java programs. We found this quite a challenge — it is much harder to poke around with low level memory management in Java than in C++ and the indeterminate nature of garbage collection made it quite hard for us to track down the problem. Eventually we discovered that we were creating a pair of socket objects for inter-process communication and one object was left blocked on a read when the underlying connection was broken.

As other articles and talks by ACCU members in recent months have pointed out, having a garbage collector does not solve all your resource problems. Well-crafted C++ can do as good a job in many cases — and has the added advantage that the C++ destructor is called at a known time in the life of the program.

Secondly, performance. Somewhat to my shock we had more performance problems with our C++ code than with our Java code. After we'd all talked about this for a while (and generated rather more heat than light) we eventually put in some instrumentation and started to measure things.

We used the Windows 2000 high resolution timer (and wrote a very simple native interface to allow us to access it from Java) to measure key parts of our program execution.

Unsurprisingly we found parsing and generating XML strings was expensive. On more investigation discovered our C++ compiler's implementation of `std::string` always increased the size of the underlying buffer by a fixed amount and so creating large strings took abnormally long times.

What options did we have?

One option was to use another standard library — STLport for example. We had a reluctance to abandon the implementation bundled with the compiler, for a variety of reasons which may or may not have been completely rational, and instead looked at optimising our XML generation.

We realised that by executing the XML generation code twice we could make it faster!

The first time through we simply counted the number of characters and didn't write anything; the second time through we actually wrote the data. In between the two calls we could allocate a string of the required size (using the "`reserve`" method) thereby guaranteeing it never had to grow.

It worked like a champ — and of course similar algorithmic changes could also improve the performance of the Java code.

Improving the algorithm was better than trying to optimise the memory allocation.

Conclusion

It was an interesting experience working interchangeably with two languages. We had a few problems which I've outlined but in general I liked the flexibility. We were able to use each language where it fitted well, and so, for example, could use 3rd party components where they were available in one language but not the other.

Although Java and C++ have similar syntax the same programs, written in the two languages, are often radically different. Fortunately they are a flexible enough pair of languages that it is possible to support a programming style which 'makes sense' to practitioners in both languages.

It is easy to make assumptions about the two languages — and it is important to be aware of this. C++ is capable of high performance, and Java does have a garbage collector — but this should not be an excuse for my laziness in assuming it will work without effort!

Roger Orr

An Introduction to 4DML

by Silas S Brown <ssb22@cam.ac.uk>

In my research project I have developed a data structure called 4DML (four-dimensional markup language), which is something like a cross between a tree and a matrix. It's not particularly efficient, but it can be an interesting way of representing things; if you can think in the appropriate paradigm, there are some tasks that can be implemented quite rapidly.

Computer Science has always used tree-like structures to represent things. Mathematical expressions are structured hierarchically, and so are computer languages, as you would know if you've ever tried to use a compiler generator. There is a vast body of theory that deals with storing and manipulating tree-like data structures for all kinds of purposes. The recent popularity of generalised markup languages such as XML is a continuation of this trend.

Much data is also matrix-like in nature. Tables and spreadsheets are matrices. A musical score (which represents parallel streams of events) is matrix-like, and so are parallel translations of literary works. Often it is possible to read a document in several different ways, using, in effect, several different methods of indexing into the items of data that make up the document. It can be useful to treat these indices as the different dimensions of a multi-dimensional matrix, so that switching from one system to another amounts to switching to a different dimension.

However, not everything makes sense as a matrix. Sometimes it is better to use a conventional tree-like hierarchical structure, particularly if the data's structure is recursive (as is the case with mathematical expressions, which can contain other expressions to any depth). But using a hierarchical structure throughout makes things more complex when they would perhaps be better represented as matrices.

The 4DML data structure aims to represent data in such a way that it can be treated as either or both, as appropriate. As a side-effect, it also allows multiple, independent hierarchies over the same data. When you read a book, you are looking at at least two independent hierarchies — the logical hierarchy of headings, paragraphs and sentences, and the physical hierarchy of pages, columns and lines. Sometimes it is necessary to store both; for example, Braille versions of books frequently have special notes to tell the blind reader where they are in the printed edition, so that, if they are in a classroom full of sighted people and the teacher says "now look at the top of the page'', they'll be able to find where this is in the Braille. Some religious texts have chapter and verse numbers, which is yet another hierarchy and can be more useful than layout (life in a diverse classroom would be so much easier if all teaching materials were numbered like that).

Why might mixing trees with matrices be useful? For one thing, navigation around the structure becomes easier to program. For example, imagine a document with several sections. Each section has several subsections, and each subsection has several paragraphs. Now imagine you have an iterator (or cursor) that is pointing to section 2, subsection 3, paragraph 6. If the document is represented as a hierarchical, tree-like structure, it should be fairly simple to move that iterator on to the next paragraph or the previous paragraph, or to go up one level and move it on to the next or previous subsection. This kind of navigation is sufficient for most conventional tasks that involve documents.

However, now suppose that the document has a translation into another language. The translation is represented as a second document, with the same structure of sections, subsections and paragraphs as the first. The iterator, as before, is in section 2, subsection 3, paragraph 6, of document 1. For some applications, it can make sense to jump from this point to the same point in document 2, that is, document 2, section 2, subsection 3, paragraph 6. If the documents can only be navigated hierarchically, then this jump will involve quite a number of navigational steps. In this particular case, the problem can be addressed by maintaining two separate iterators (one in each document), but things could get more complex.

For example, imagine a music book with many songs. Each song has many translations, each translation has several verses, and each verse has many words. Each song also has music, which has several parts for different instruments. The whole thing can be represented hierarchically, but it would be a fairly complex task to jump from (song 3, translation 1, verse 2, note 5) to the same place in translation 2, or to the same place in verse 3, or to the data that represents which notes are being played at that time. Such jumps would be required very frequently by certain kinds of typesetting algorithms.

Problems like the above arise because some hierarchical representations are not "simply connected". In a tree-like data structure, there may be no direct link from, for example, (song 3, translation 1, verse 2, note 5) to the same place in translation 2, even though there is a conceptual link as far as some types of processing are concerned. If conceptual links are not physically represented in the data structure, then the algorithms that use them will become more complex and difficult to develop, since there is a more considerable code overhead for navigating around the data.

The problem could be addressed by adding more links (pointers) in the hierarchical structure, but doing this can become cumbersome. However, adding more dimensions can allievate the need for such pointers, because the items that are linked are adjacent to each other in a different dimension. (Of course, the actual implementation might use references or pointers, but the programmer doesn't have to think about that.)

To illustrate: take a piece of paper, mark any two points on it, and then try to fold the paper in such a way that those two points touch each other. In two-dimensional space (on the piece of paper), those two points are quite far apart, but in our 3D space, they are touching. If you were trying to explain all this to imaginary beings who can only understand two dimensions, then you would have to say that there is some kind of "wormhole" magically linking the two points together. But if you can think in three dimensions, you don't have to invent such things as wormholes — the points are touching because the paper is folded, and they are adjacent to each other in the third dimension. Mathematically, the space on the paper used to be "simply connected" (each place on the paper was only "touching" the places next to it), but when the paper was folded, it was no longer simply connected (some places were now touching other places that were quite far away on the paper). However, in three dimensions, it is still simply connected.

Then why not work with N-dimensional matrices directly?

Computer memory is one-dimensional; everything is indexed by memory address. It is possible to represent objects of two or more dimensions in computer memory, but doing so requires somebody (either the programmer or the compiler writer) to think about how to store such things in a one-dimensional space. This can be particularly difficult if there is no limit to how the object can grow. Anybody who has tried to write a simple full-screen text editor from scratch will testify that this is not as easy as a novice might imagine; as the user moves the cursor on a two-dimensional screen, the editor has to calculate where it is in the one-dimensional sequence of characters that makes up the file being edited.

This problem is made more acute when the number of dimensions is not constant. Additionally, when representing a document in an N-dimensional matrix, the resulting matrix is sparse; most of the data is close to the origin, and it would be very wasteful to allocate a large N-dimensional array that is big enough to hold the maximum extent of every dimension.

There are many ways of representing sparse N-dimensional matrices in computer memory. 4DML is one of these, but it has been designed in such a way that it can also represent hierarchical structures, or a mixture of the two. To begin with, we note that an N-dimensional matrix can be represented as a set of three-dimensional points:

(dimension, position, object)

In this representation, each object (which we assume can be uniquely identified) is given a position on each of the dimensions. For example, if object Q is at co-ordinates (5,8,13,21,34,55) then the three-dimensional point-set will contain the following points (in no particular order):

(1, 5, Q), (2, 8, Q), (3, 13, Q), (4, 21, Q), (5, 34, Q), (6, 55, Q)

In a real application, it is useful to give the values of the first co-ordinate names rather than numbers; 4DML terms them "elements", since they often reflect the names of XML elements, and it is important not to confuse a dimension in the N-dimensional matrix with a dimension of the 3D representation.

The fourth dimension in 4DML arises from the need to also represent hierarchical structures. It is a depth dimension, and it is used to disambiguate elements with the same name at different depths in the tree.

[concluded at foot of next page]

ACCU UK Python Users Group

"The sound of one coconut shell clopping"

by Paul Brian <paul1brian@yahoo.com>

Welcome to the Python section of C Vu.

Python is a relatively new programming language that has been gathering momentum and popularity in recent years. This section of the magazine, like the ACCU-SIG "PyUK" (the Python UK Users Group) that its authors spring from, is dedicated to spreading the word, or at least trying to point out the benefits of a surprisingly elegant, simple and powerful language that has long outgrown its "scripting only" designation.

Python is used in such diverse places as NASA, Industrial Light and Magic (the Star Wars special effects people), Disney, Yahoo, Google, Thwate, RealNetworks and even IBM. It can run on a huge diversity of OS/hardware platforms (32 at last count, personally I am waiting for the ZXspectrum port) and can call and be called from Java, C and C++ code.

In short, Python is a object-orientated, very high level programming language with surprisingly clear syntax, extensive built-in data structures, strong but dynamic typing and has an enormous number of modules and libraries covering some useful, abstract and occasionally strange capabilities.

Python is sometimes characterised as everyone's favourite second language, and I would encourage anyone involved in C, C++ and Java systems work to look very closely at its capabilities, especially its ability to work seamlessly with those languages.

Anyway, some background to Python and a little introduction...

Origins of Python

One Christmas, about a dozen years ago, a young man called Guido van Rossum sat down and wrote an interpreter for a new language he wanted to create as an improvement on a previous effort. This language became Python (and yes, it was named for John Cleese and friends). In keeping with Python's offbeat history and great portability, it first ran on an Apple Mac.

Python was released from the Amsterdam laboratories where Van Rossum worked in 1991, and has been growing in popularity ever since. There are now an estimated 1/2 million programmers, 40+ books on Python and innumerable articles (now plus one).

Python is open source and free.

The Python community

Python has a very strong community feel, centred around the `comp.lang.python` newsgroup. The group is refreshingly free from flames, egos and off-topic blathering, even though you can still find posts on how to grow certain anatomical parts by 3 inches. That apart, the group is a sadly rare thing these days, a relaxed, open, collegial group that offers help, advice and support to all from greenest newbies to the most experienced. And it is done in a generally light-hearted tone. Just like the ACCU :-)

However, Python itself takes programming seriously, even if "Pythonistas" do not take themselves too seriously. No, I do not know why Python users are called Pythonistas. It's just one of those things.

[concluded on next page]

Thus 4DML is a collection of points of the form:

(element, position, depth, object)

These points can represent any hierarchical data structure and also any N-dimensional matrix, or a combination of the two. They can also represent several different structures over the same data. The ordering of the points themselves is not significant, which means they can be kept in a hash table or other efficient data structure.

Transforming it

An algorithm called "transformation by model" takes two 4DML documents, an "input" and a "model", and transforms the input as necessary to reflect the structure of the model. The algorithm can also report which objects have been lost in the process, if any.

The algorithm works by performing a top-down traversal of the model, and reads off relevant parts of the input as it does so. Since 4DML can be read in many different ways, it is not difficult to read it in whatever way is dictated by the structure of the model, regardless of whether or not this matches the original structure. Hence it is possible to perform complex structural transformations merely by writing down the form of the desired result. Of course, it is also possible to follow the structure of the input like a conventional stylesheet processor.

I won't go into the precise workings of the algorithm here (it would take too long) but readers are welcome to download and experiment with the hacky Python prototype on my website, at `http://www.cus.cam.ac.uk/~ssb22/4dml/`. The prototype can take its input in XML as well as a few simple text-based languages specially invented for the purpose (these are usually easier to hand-code than XML is). It is only a proof of concept though; it can be used, but it would need more development before it is a quality software deliverable (but then, academic research is not about making software deliverables).

Of course, feedback is most welcome, although I cannot guarantee to reply to all of your hundreds of emails :-)

Silas S Brown

Python is kept alive and changing via an open process (Python Enhancement Proposals (PEPs)) that discuss what improvements could be made. The final decision is Van Rossums', but it is a sort of Chinese parliament. Van Rossum and a core group of developers keep up an impressive stream of improvements and changes that have satisfied just about everyone, so it seems to work.

There are also many SIGs (including the UK Python Users Group – an ACCU SIG) and quite a lively conference and training scene.

Using Python

To get a copy of Python please visit `www.python.org/2.2.1/`. This has just about everything you want. Most of us have worked through Guido's tutorial that is included.

Windows users might find the release at activestate.org more suitable (it includes some win32 extensions that make life in Windows much easier. Automating Word and Excel has never been [so] much fun.)

Amazingly the python interpreter has also been rewritten in Java. This means python code can be run under a JRE, allowing python code to call and be called from Java code. If you want to play with this try `www.jython.org`.

And so just to show what Python can do, (but do not hold it responsible for this programmer's failings) here is a piece of code that downloads a web page, parses it and emails that day's news headlines to the author in less than a dozen lines of code.

```
import urllib, smtplib           #Pythons vast library store is mostly text files of code added onto a path.
headlines, en, y = [], 0 , len('<B class="h1">') #setup a list to store headlines, and a couple of useful bits

htmlFile = urllib.urlopen('http://news.bbc.co.uk').read() #visit the BBC site and get the html as a single string

while 1:                                        #an infinite loop (break condition lower down)
    st = htmlFile.find('<B class="h1">',en)     #find beginning of each headline
    en = htmlFile.find('</B>', st)              #find end of each headline
    if st == -1: break                          #if no more headlines stop
    headlines.append( htmlFile[st + y:en] )     #slice up the html string at the start and end of the headline,
s = smtplib.SMTP('post.myServer.co.uk')         #open up a connection to smtp server
s.sendmail('Python@myServer.co.uk',             #send the message (From, to and body, using string formatting)
           'pbrian@myServer.co.uk',
           'Subject:Daily News\r\n\r\nThe headlines are:\n %s \n %s \n %s' %
           (headlines[0],headlines[1],headlines[2]) )
s.quit()
```

The first thing you probably notice is the lack of braces or brackets. Whitespace and indentation are actually part of the syntax in Python. After a while it becomes natural – honestly, try it for a few hours. Eric Raymond's comments on this follow most of our own experiences (see below).

The first line imports some helper libraries. There is a plain text file on my machine called `urllib.py`. I can open it and read the python code. It uses some clever socket manipulation to run HTTP to the BBC site and get me my news page. That library is part of the standard distribution. Python gets referred to as "batteries included". This is why.

Where are the type declarations? Well, Python discovers type at run-time, meaning there is strong typing (you cannot conjoin integer 12 and string "3" to get 15 (or 123), but it is dynamic, allowing less lines, less breaks in the flow. I like it.

And finally the data structures – lists are simple to set up and easy to manipulate. Dictionaries or associative arrays are just as easy (they use curly brackets not square) and they can hold anything. A dictionary containing a list of strings, dictionaries and other lists is easy. Pointless, perhaps, but easy.

Python can stretch from a 10 line throwaway to hundreds of thousands of lines of code, run on a huge variety of platforms, interact with a huge variety of other software whilst still retaining clear, readable and straight-forward code. This deserves an explanation. My best attempt is that Python takes programming seriously. This means that it is not designed for a given application domain, nor built with an eye on 30 years of backwards compatibility or the marketing needs of a multinational.

I have read that "Python gets its compromises right". I think it does too. Give it a try: you might like it :-)

Enjoy!

Paul Brian

Useful URLS

"Official" Distribution: `http://www.python.org/2.2.1`

Windows, with COM access: `http://activestate.com/Products/Download/Get.plex?id=ActivePython`

Tutorial: `http://www.python.org/doc/current/tut/tut.html`

Origins of Python: `http://www.python.org/doc/essays/foreword.html`

PEPs: `http://www.python.org/peps/`

Jython - using Python and Java: `http://www.jython.org/`

SWIG - using C, C++ and Python: `http://www.swig.org/`

Introductory Articles: `http://www.python.org/doc/Intros.html`

Interesting comments from Eric S Raymond: `http://www.linuxjournal.com/article.php?sid=3882`
`http://europython.zope.nl/interviews/entries/eric_raymond`

Comparisons to other languages: `http://www.ipd.uka.de/~prechelt/Biblio/jccpprtTR.pdf`
`http://home.pacbell.net/ouster/scripting.html`

Number of programmers: `http://www.activestate.com/Corporate/Communications/Releases/Press957675912.html`

Who else uses Python: `http://www.python.org/psa/Users.html`

Reviews

Bookcase

Collated by Michael Minihane
<michaelm@pobox.co.uk>

Francis Glassborow writes:

As a sort of interesting follow up to my comments last issue about Addison-Wesley they have asked ACCU members to help them get it right. If you look at the letters pages this issue you will find a letter from them asking members to help with the next edition of Stan Lippman's *C++ Primer*. I would be saddened if you do not grab this chance to make a difference. It is important to help publishers, which is one reason why many of you spend time reviewing books both before and after publication. (And we still need more hands to share the work, so if you are not already contributing to this column perhaps it is about time you dipped your toe in the water.)

Please remember to mention our reviews when visiting shops. It may not be obvious but sources of good advice on specialist books is nearly always welcome. It is just one more way that the ACCU brand can get wider recognition.

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
The PC Bookshop (020 7831 0022)
orders@pcbooks.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 some books shows the dollar pound conversion rate where known. I consider a rate of 1.5 or better as appropriate (in a context where the true rate hovers around 1.6). I consider any rate below 1.3 as being sufficiently poor to merit complaint to the publisher.

Java

Java Cookbook by Ian Darwin (0 596 00170 3), O'Reilly, 850pp @ £31-95(1.41)
reviewed by John McLaughlin

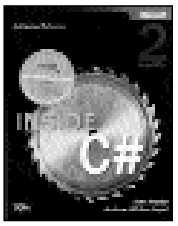
As far as extended metaphors go, the 'cookbook' should be a familiar terrain to most. What you get with Java Cookbook is a twenty-six chapter smorgasbord comprised of both the staple and more exotic ingredients of the core and non-core Java APIs.

Relative newcomers to Java will probably find much in this book that leads them to the correct place at the table, even ordering their meal for them. Intermediates, could well experience a deepening of the palette, suffering only the occasional irritation from the warmed up remnants of previous meals. More advanced users are bound to find something in the general dinner conversation that is teeth grindingly annoying, should they care to look for it. This is not meant as a criticism. As an exercise in hand-holding this book surpasses itself, leading the reader through the APIs with cool assurance, each chapter following a rigid format: an introductory talk followed by the 'recipes' themselves, which are divided into three sections; Problem, Solution and Discussion.

Throughout the book is immensely readable and at pains to clarify rather than impress with the sound of its own voice. As a desktop reference, however, it may well already be somewhat out of date, stopping short, as it does, of Java 1.4.

Nevertheless, a clear, bright piece of work, bursting at the seams with ideas for connoisseurs and snackers alike.

C# & .NET



Inside C# by Tom Archer & Andrew Whitechapel (0 7356 1648 5), Microsoft*, 872pp + CD @ £36-99 (1.35)
reviewed by Francis Glassborow

I think the first thing to clarify is what this book is not. It is not a book for programming virgins, nor is it a book for those that do not already have at least an implicit understanding of object orientation. It is also not a book about programming for .NET nor is it a book about the way that C# has been designed. I think that last point is particularly important because many readers have an expectation that books with this kind of title will address the programming model that a language uses.

So what is this book? Well it is a fairly well written introduction to the C# language for those who are already reasonably competent with at least one other object oriented language. Those that already code in Java or C++ will find the material reasonably presented though a few more comments on the ways that C# differs from these popular languages would be helpful.

However where I completely part company with the author is in his example source code. Let me take just a single example. On pages 320-326 he purports to demonstrate how essential `goto` is to writing source code. Now I will pass over his use of switch statements where any well trained user of an object-oriented language would wonder why runtime polymorphism has not been used because maybe he feels his reader will not be happy with having to deal with the subtleties of 'virtual construction' and simply say that in each of his examples I can think of a clean and simple way of writing the code without using `goto`. Please note that I am not rabidly anti use of `goto`, what does concern me is that so many programmers use it because they do not understand how to write code where its use is unnecessary.

Now, if you are happy to ignore the quirkiness of the author's code and be happy with learning the C# language in isolation then this book is acceptable. However please do not let weak programmers near it because it will only serve to re-enforce their bad habits. Also avoid letting novices anywhere near it.

Learning the syntax and basic semantics of a language is fairly easy; learning how to use a language to express thoughts and solutions clearly and correctly is more demanding. It takes time for languages to develop their own idioms that encapsulate good practice. It would be unreasonable to expect authors to understand or invent good idioms straight away. However they should not be repeating poor coding styles from other languages.

Microsoft C# Programming for the Absolute Beginner by Andy Harris (1 931841 16 0), Premier*, 475pp + CD @ £21-99 (1.36)
reviewed by Francis Glassborow

There is one very big drawback to this book and you will find it on the back cover where it says 'Readers must have the .NET Framework and Microsoft Visual Studio in order to run the CD.' Rephrase that to say that readers must have those products to use this book. Currently that is a pretty big up front investment for a novice programmer. In fact that is the biggest barrier to many who would like to learn C# and makes languages that provide free implementations much more attractive to novices. While the authors cannot do anything about this problem I hope Microsoft will address it in the very near future.

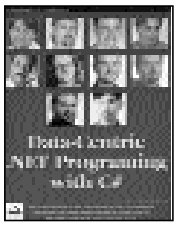
Now the style of this book (along with other books on programming languages in this series) is very much addressed at the novice enthusiast. The underlying theme is to learn programming by writing programs for games. I think this concept is flawed for two reasons. The first is that many people want to learn to program that have no interest in games and do not find the subject enthusing. Actually quite a few games players would find the kind of simple-minded game that books such as this have to use something of a turn-off. The second flaw is, in my mind, much more serious; this kind of book is written for a male mind-set. Programming is already far too dominated by one half of the human race and books such as this one will just make it worse. Anywhere else such a blatantly sexist approach to a technical subject would receive immediate condemnation. Now that this book has made me recognize this problem, I have to wonder how many authors of books for novices ever consider whether their choice of exercise material is strongly biased towards a single sex.

The language coverage and general programming is perfectly reasonable though, in my opinion only suitable for young enthusiasts. If you already know how to program, or you

wish to learn to program to professional standards you will need to look elsewhere. I think another flaw in the design of the book is that it focuses on instant gratification at the price of understanding. It would be easy to work through this book without actually writing a single line of original code. Worse, as we know, an important element of learning to program is learning about solving problems and that takes practice.

I think that a book aiming to teach that lacks any exercises is missing a vital element. Granted that each chapter ends with a set of challenges, however none of those is exactly demanding. I have a distinct feeling that the author is more interested in providing readers with a sense of satisfaction than with actually making them think. I note that he manages a Streaming Media Lab in a University Computer Science Department. That leads me to wonder if his view of programming is just as a tool for students working on streaming media.

In conclusion, this book is technically satisfactory for its target readership but I think that its style leaves more than a few questions. If you want to learn about programming there are better books. If you want to learn C# there are much better books but if you want to try your hand and find simple games programming interesting and do not mind being spoon-fed the answers this would be the book for you.



Data-Centric .NET Programming with C# by various (1 861005 92 X), WROX*, 784pp @ £43-99(1.36)
reviewed by Asad Altimeemy

This book is about using .NET technologies to work with data. Since data can reside in different forms at different places, this book focuses on; databases, XML/XSL, directory services and files.

If you are a programmer with extensive experience with C# and ADO.NET this book might not be very helpful. However, if you' are doing database programming with .NET and still learning ADO.NET and C#, then this book is a very good choice.

It has a good spectrum of coverage and is really for those applying C# and .Net to a real world situation. It is not going to teach you syntax for C# but will show you how to use the .NET framework for Data intensive apps using technologies such as XML and Active Directory. In addition it discusses how to migrate existing apps over to .NET.

All in all this is a very good book. I would give this book 8/10 points and recommend it to any developer seriously interested in .NET C# programming.

A word of warning – this book has been written with the Beta 2 version.

Special Edition Using C# by NIIT (0 7897 2575 4), Que, 564pp @ £36-50 (1.37)
reviewed by Asad Altimeemy

It's always worrying when you get a book without 'official' authors. It looks like the authors are too embarrassed to put their names on the cover of the book.

The book starts with the C# story, comparing C# to C++, Java, JScript and Visual Basic. Then there is a superficial introduction to .NET and programming in C#. We are told nothing with substance in areas like *Delegates* and *Events* or *Exceptions*. The reader feels rushed, hurriedly going from one topic to another, without the benefit of good examples or even a discussion on how you would use this new programming language. There are only 20 pages on debugging and 25 pages on Windows forms. There is very little coverage of ADO.NET, Web services and ASP.NET. It is actually meant to be a book on using C# with the above technologies!

In the introductory section of this book and repeated under the subheading is the phrase 'This book is for you'. It is targeted at programmers who want an in-depth coverage of C# on the .NET platform. Having read this book I must disagree with their opening phrase. For in depth knowledge you will definitely need more coverage and more discussion than is covered in this text. We are only taught how to use C# to develop simple projects. The book would only be suitable for beginners to C# with some knowledge in C++ or Java programming. The text fails to cover any topic in reasonable enough depth for the reader to achieve some sense of understanding and knowledge of C# and the .NET.

Methodologies & Process

Agile Software Development by Alistair Cockburn (0 201 69969 9), Addison-Wesley, 278pp @ £26-99 (1.30)
reviewed by Chris Czarnecki

Aimed at the experienced audience this book brings a refreshingly different and pragmatic approach to software development. The Agile processes question the traditional process intensive approaches to software development and asks how they can be changed to provide the shortest, simplest route to successful software development.

Agile software processes are built on four core values:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

The important thing about this book is that it recognises that all methodologies have limitations and does not detail one in particular but emphasises that they have to be tuned to not only the project but the individuals in the project team.

The first three chapters are rather abstract but introduce important concepts and also stimulate an alternative way of thinking about software development – as a co-operative game of invention, with the ultimate goal to win whilst always remembering what they have learned along the way. Communication and errors in communication form the core of chapter one, whilst chapter two concentrates on people skills and how different skill levels can be integrated and compensated for. Chapter three considers the effects of the physical environment and the cultures of the development teams. Chapters four and five are the really important chapters of the book. They detail what a methodology is and the principles used in designing and evaluating them. eXtreme programming is overviewed as an example of an agile methodology. Chapter five considers a technique for evolving a light-but-sufficient, project-personal methodology that can be useful to the project. Chapter six describes the author's own family of methodologies (Crystal), along with the principles for tuning them to meet individual project and teams needs.

The presentation style of the book is extremely clear and it is written with authority. The author has obviously worked in this area for a number of years and has drawn upon his experiences and research to provide first class examples throughout. There is no doubt in my mind that this is the most useful software process book that I have read for a number of years. Whilst the material described may not be totally suitable for all projects, everybody involved in the software development industry, from IT executives to junior programmers should read it.

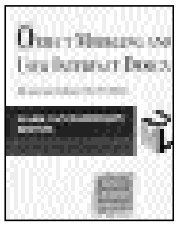
Java Tools for eXtreme Programming by Hightower & Lesiecki (0 471 20708 X), Wiley, 515pp @ £29-95 (1.34)
reviewed by Chris Czarnecki

Anybody who needs help with automating the testing and integration of their Java software can benefit from reading and using the tools and techniques presented in this book. Whilst the title includes eXtreme programming it is misleading to think that the material is only appropriate for those practising eXtreme programming. Really the book provides an excellent tutorial on open source tools for automating the testing and integration of Java software.

The book begins with an introduction to the twelve core practices of eXtreme programming followed by an overview of J2EE deployment concepts, in particular web application WAR files, Enterprise Bean JAR files and J2EE EAR files. The tools are introduced by way of a case study - an online pet store. All the example code can be downloaded from the book's web site. The first tool to receive cover is Ant, a build tool that enables the user to automate the build process. Importantly it is highlighted that Ant compliments IDEs rather than replacing them. The three chapters on Ant are excellent, introducing the basics and then building up to a full J2EE project, including servlets, JSPs and EJBs. Unit level testing using JUnit is the next subject covered. The features and facilities are covered and its integration with Ant detailed. Unit testing on Servlets and JSPs is undertaken using Cactus, which allows in-container testing

to be performed. For functional testing of web based applications HttpUnit is detailed. Finally, performance testing for applications is discussed with JMeter and JUnitPerf. These are complimentary tools, which if used correctly allow potential bottlenecks to be highlighted early in the development process.

All the tools described in this book are freely available and of high quality. However, as is often the case the accompanying documentation is not so detailed or missing. This book provides such documentation and makes using these tools much easier. For anybody wanting to improve their build and testing processes I strongly recommend this book.



Object Modelling and User Interface Design by Mark van Harmelen (ed) (0 201 65789 9), Addison-Wesley*, 452pp @ £34-99 (1.28)
reviewed by Paul S Usowicz

When I received this book I was very excited as I have a keen interest in both UML and user-interface design, having spent many years designing GUIs for industrial machinery using C++ and UML. My excitement soon turned to disappointment when I realised that this book was not what I had been hoping for. I was hoping for a nice structured walkthrough of developing user interfaces with UML. What I was presented with instead were 10 chapters by different authors with varying amounts of theory and examples.

I found the book very difficult to read and did not actually finish several of the chapters because they were either extremely hard going or a thinly disguised advert for a particular product. If I buy a book to learn how to use UML for a specific task I do not expect to then have to buy a specific application when I already have numerous UML editors.

The latter chapters in the book (especially chapter 9 – Toward Unified Models in User-Centred and Object-Oriented Design) were much better with quite a few nuggets of information gleaned.

The book is definitely written for someone after academic, as apposed to hands-on, information. Users, or potential users, of particular products (Idiom, OVID) may find it more interesting than most with various product comparisons scattered throughout.

Would I buy the book? No. I was expecting tools that I could use on a day-to-day basis but instead received some well written but, for me, mostly irrelevant information. Please bear this in mind if thinking of purchasing the book, especially if you are after a more academic read.

The Joy of Patterns by Brandon Goldfedder (0 201 65759 7), Addison-Wesley, 176pp @ £22-99 (1.58)
reviewed by Francis Glassborow

Let me start by highlighting that this is an even smaller book than might be suggested by the page count. The appendices start on page 111 and those are almost entirely source code that add nothing by being printed – they should have been on a CD. Having said that let me look at what is left. The authors intent seems to be to walk the reader through a number of very simple design problems and illustrate how patterns can help. I think the sub-title– Using Patterns for Enterprise Development– is grandiose to the point of deception. This is a simple little book that may help relatively inexperienced readers to get some understanding of what patterns are and how they can help. Unlike one of the reviewers on `www.amazon.co.uk`, I cannot agree that the price is cheap, even in comparison with other books on programming. Yes the author writes clearly and has produced a good introduction to the idea and gives examples in several languages but £22-99 is not cheap, and even the discounted price of £18-99 is rather expensive for a 110 page introduction padded out with 60 extra pages of source code. Pearson Educational really must address its pricing policy.

Building Effective Project Teams by Robert Wysocki (0 471 01392 7), Wiley, 276pp + CD @ £33-50 (1.34)
reviewed by Joe Mc Cool

There is an air of unreality about this book. I can't help but feel that its background stems from deep within academia, perhaps out of touch with the real world.

Note the title; it is about building a project team. It is about human beings, recognising their strengths and weaknesses. It is about project type recognition and choosing the personality profiles to suit. If one is to believe the current author, all one has to do is construct a few scatter graphs or Venn-type diagrams and the job is done. I'd guess that 20% of this book's paper usage is taken up with such graphics.

Of course, it's true, we want to recognise the good team player, the gifted, the industrious, but no amount of scribbling on paper will give us that. That is one of life's dilemmas and judging from my own experience (which is considerable) it's so very easy to be wrong. I find that the good team players are recognised most easily outside office hours. It's when one goes camping with them or sailing that their real personalities show. The question then is one of creating opportunities where true worth can be recognised. How does one probe? What questions should one ask? What are the useful exercises? Wysocki is not addressing that level at all. He seems to suggest that if the theory is in place (illustrated graphically of course), then the practice will follow.

It is rare for a project manager to able to pick at all. Most times he has to make do with the motley crew he's got trust upon him and I would suggest that the current volume would give him little solace. To browse in the library perhaps (for some of the buzz terms), but to spend good hard earned cash on, no.

Web Tools

Automated Web Testing Toolkit by Diane Stottlemyer (0 471 41435 2), Wiley, 285pp + CD @ £35.95 (1.39)
reviewed by Christopher Hill

The sub-title reads 'Expert Methods for Testing and Managing Web Applications'. I was really looking forward to learning a great deal about this very neglected area and was greatly disappointed in this book.

The author is a 'Certified software test engineer' and this book reads like a 1980s Quality Assurance manual, which has been reworked (a little) to weave in the latest languages and techniques. The book has the feel of a number of different 'seminars' bolted together, in no particular order. Business Requirements and Testing Methodology (Unit Testing, System Testing, Functional-black box testing. Regression testing and user acceptance) are 'covered' to a greater depth, but in direct proportion to how many paper based forms and templates are required. Life Cycle Models (Waterfall, V-shaped, Prototyping, Incremental, Spiral) are 'covered' in a paragraph each, towards the rear of the book. If you need this sort of information see the Wiley book *Testing Computer Software* (Kaner).

As an aside ADA is listed as one of the languages that would be amenable to Unit Testing – while this might be the case how does this help the web site implementer?

There are two lists of web-related tools and resources. Each entry has a URL and a very short paragraph. Like everything else in this book a stepping-stone to somewhere more profitable. One whole chapter is given over to testing of different platforms and servers. In practice this is a page précis for each product listed (the ZD Net web site receives a credit for the information). Another Wiley book *Testing Applications on the Web* (Nguyen) provides much better coverage and detail of this area.

To sum up, a dose of 1980s QA; out of date search results for web tools and resources; lots of paper based forms and templates; very poorly structured, touching on material much better covered in other Wiley books. Not recommended.

Platform Specific

Programming Windows Services by Randy Morin (0 471 38576 X), Wiley, 428pp + CD @ £35-95 (1.39)
reviewed by David Nash

The subtitle of this book is 'Implementing Application Servers' by which the author means programs designed to run continuously, responding to calls from clients. What the Unix user would call daemons, in fact and he does point this out – even giving the etymology of the word! In the Windows world such things are usually implemented as NT Services or, now that we are several versions on from NT, simply Windows Services as the title indicates.

So can you fill a book on the subject of how to write a Windows Service? Probably not, but

this book doesn't try to. Only the first 2 chapters are devoted to the technical details of how to write and use them. The rest of the book covers a diverse range of topics useful to the writer of these services including inter-process communications (IPC), the Windows event log, configuration and security.

The chapter on IPC is very wide-ranging, covering such things as socket communications, CORBA and of course (in a Windows book) the component object model, or COM. Because of this each subject only gets basic coverage. There are in fact other whole books twice as thick as this one on single topics from that chapter so don't expect comprehensive details, but the introduction to each is good enough to give you a flavour of what it's all about and whether or not you want to use it. The section on COM in particular scores points from me by explaining what is really going on and not just relying on the automatically generated code from the Visual Studio Wizards.

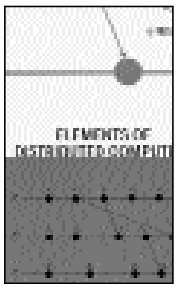
The book comes with a CD-ROM containing the sample code, which is scattered throughout the book and is generally of good quality. The author is a firm believer in using C++ in the way it was intended and is not scared to use such things as namespaces, exceptions and the standard library, only using Windows-specific types (DWORD, for example) when a call to the Windows API demands it.

The examples show how to use specific features and in several cases a generic framework is provided for the reader to use when implementing his/her own application. For example, a C++ base class is given from which your own class can be derived, thereby implementing a Windows Service simply by overriding a couple of methods.

If I had any criticism it would be simply that the book does not go into enough depth on many of the subjects covered – both core Windows services and some of the associated technologies. The SRVANY utility, for example, which comes with the NT resource kit, gets one sentence that I will quote it in full here; '*The SRVANY command-line utility can be used to convert any executable into a service*'. There is no mention of where to get it, or the pros and cons and possible pitfalls of using it.

Overall a good book, clear and written in an easy to read style and to be preferred over simply choosing NT Service on the ATL AppWizard if you have to write a windows service and don't know where to begin. Recommended.

Distributed Programming



Elements of Distributed Computing by Vijay Garg (0 471 03600 5), Wiley, 423pp @ £55-95 (1.34
reviewed by Francis Glassborow

The subject of this book is becoming increasingly important to practitioners as well as academics. Of course the practitioner can take a purely pragmatic view of learning about the subject. That will work fine until they have to sort out some problem. Recently I came across a school with a network of over 500 computers and with plans to increase that to over a thousand. They had eventually come to realize that they had serious problems with the school's IT infra structure but had no idea as to what to do. Unlike some, they were willing to go and find someone with appropriate understanding who would manage it. They had enough insight to recognize that they would have to look outside education. Of course one of their problems was that they were running an entirely flat LAN. Any competent network administrator would recognize the problem and some might even understand that it was a structural problem that is solved by knowledge rather than by buying lots of new hardware.

Though this book is not relevant to that problem there is a certain similarity. Many acquire pragmatic solutions to problems that are fine for the small distributed computing arenas where they start. However it rarely stays at that level and eventually understanding of the basics becomes essential. This book tries to address this need among practitioners as well as amongst more advanced students (first year graduate level of there abouts). I have no doubt that the books approach works well on the academic side, particularly as it has a rather fuller view of the problems than many of the alternatives. The author has tried to choose examples that at least have some basis in practical applications.

So why am I so dubious? Well reading the book requires some fluency in both a Pascal like pseudo source code and the symbolism of mathematical logic. While most programmers can easily mange the former, the latter is a step to far for most.

If you are not phased by formal logical expressions and want to get a grip on the more theoretical underpinnings to distributed computing this would be a good book to study. However, be warned, you are going to have to work hard. It would probably be an advantage to have a mentor to help you check your understanding as well as to cope when the going gets tough.

Quick CORBA 3 by Jon Siegel (0 471 38935 8), Wiley, 372pp @ £28-95 (1.38)
reviewed by Dave Nash

CORBA 3 is the latest of the Object Management Group's recommendations for the common object request broker architecture, although the various new facilities have been trickling in via point-releases since as long ago as CORBA 2.2 in 1999. Quick CORBA 3 is a book that explains what these new facilities are and how they can be used. The book doesn't cover CORBA basics, but rather assumes that the reader is already familiar with some previous version of CORBA (At least 2.0) and the author recommends his previous book, which covers the basics of CORBA (including version 3), for those who need it.

The new facilities provided by CORBA 3 and covered by this book include Java integration and value types, the XML mapping (which uses value types to implement the XML Document Object Model API), the interoperable naming service, asynchronous messaging using callbacks and store-and-forward facilities, Quality of Service provisions, real-time, fault-tolerant and minimum varieties of CORBA, CORBA Component Model and finally modelling with UML and CORBA metadata facilities (including a couple of chapters by other members of the Object Management Group).

Although there are descriptions of IDL and the author has included code samples throughout, the book does not really get into a lot of technical detail, concentrating more on describing the concepts involved. This makes the book more suitable for development managers or architects than for programmers as a reference – which it really is not.

Because of this, as a programmer, I found the book a little wordy and found myself thinking 'where is the code?' on a number of occasions. Although if you need to understand the new facilities provided by CORBA 3 without having to dive into the code, this book provides a good overview. If you are looking for a programming reference, look elsewhere.

Mastering RMI by Rickard Oberg (0 471 38940 4), Wiley & Sons, 302pp + CD @ £35-95 (1.39)
reviewed by Peter S Tillier

As I work in a middleware development environment I was interested in reviewing this book for two reasons; a) its relevance to my job – expanding my knowledge and b) the recent RMI technology issues and changes. I have used some simple RMI code using Java and C when I worked as a trainer in my previous job.

The book is supplied with a CD, sadly, the one that I received was cracked in transit and I would not risk running it in any of my CD drives - IMO there is too much risk of damage to the drive unit if the CD breaks up. What is annoying is that I sent two emails to Wiley asking for a replacement, without success. The first resulted in an automated response referring me to another email address to which I forwarded my second request – and I'm still waiting for a response. Very, very poor customer service Wiley! So I cannot comment on the CD contents at all.

I think that the book, itself, is not well thought-out or laid-out. The chapters cover the topics, but not in a very readable way. In technical books, I think that layout is very important and this book often intersperses code with commentary on the code (not in itself a bad idea), in what seems to me to be a very sloppy way. This often results in code being mixed with commentary and pictures or diagrams. In one case a side-bar is included in the middle of a 'tip', which is itself inside normal text – if I were the author I'd be extremely disappointed with the way in which my work has been presented. Further examples of this lack of attention to layout extends to the code listings, which in many cases are not split

in the most appropriate way across page breaks.

The first two chapters are actually quite well laid-out and describe client-server applications and the use of RMI in a reasonable way, but the later chapters exhibit the problems that I mentioned earlier. It is almost as if the book has been produced to very tight time-scales and the layout and proof reading, which, in my opinion, ought to have corrected some of the layout problems, were given little consideration.

The book is in four sections. An Introduction, then a section describing simple RMI applications and the use of sockets and activation. The third section is devoted to some more complex applications - a chat application and a mobile agent. The last covers the use of RMI with JINI and EJB.

I would have enjoyed reading this book much more if the layout had been better, so I cannot recommend it in its current form. Others may take a less critical view of the layout than I and so be more generous, but for UKP 34 I think that readers are entitled to expect more.

Java RMI by Willam Grosso (1 56592 452 5), O'Reilly, 544pp @ £28-50 (1.40)
reviewed by Emma Willis

This book assumes little knowledge of distributed programming or networking so introduces the vital concepts before teaching the fundamentals of the remote method invocation (RMI) framework. It is targeted at competent Java developers.

The first section follows two examples to practise use of the RMI methods – a basic printer server and a more complex banking server with ATM application. I was impressed with the chapter that compared the RMI and non-RMI printer server examples; it highlighted the simplicity of RMI in comparison to solely using the java.net tools.

Unfortunately, examples are limited to code excerpts. At least towards the beginning of the book I felt complete examples were necessary to introduce Streams and Sockets. Although you can download the complete code from the associated website, I felt that pages did not make clear which Java packages were needed to produce the code.

The book is not only focused on coding. It addresses system design and software-engineering issues when first designing distributed applications.

The second section of the book deals with scalability issues, looking particularly at how the serialisation and Thread capabilities of Java can be applied to RMI applications.

The real differences that the RMI framework can make are not introduced until the final section, targeted at the more advanced programmer. The aim here is to introduce the juicy elements that makes Java RMI so impressive, so useful and so flexible. It looks at the Registry naming service, custom sockets and dynamic classloading.

Finally, the book addresses some of the security issues involved in distributing applications. It suggests design patterns – factories and activation frameworks. Then, almost as an afterthought, the author introduces CORBA and suggests how RMI and CORBA can be used alongside each other. I think this final section appears hurried and adds little to this book.

get in the groove by Phil Stanhope (0 7645 4893 X), Hungry Minds (Wiley), 450pp +CD @ £37-50 (1.33)
reviewed by Francis Glassborow

[see web]

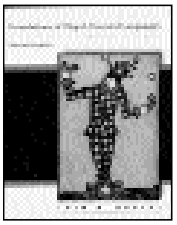
Peer-to-Peer Programming in Groove (0 672 32332 X), Addison-Wesley, 482pp + CD @ £28-99 (1.38)
reviewed by Francis Glassborow

[see web]

Other Programming

An introduction to Ada, 2ed (revised) edition, 1984, by Stephen John Young, (0-85312-804-9) Ellis Horwood, (0-470-20112-6) Halsted Press, 336pp @ unknown
reviewed by Colin Paul Gloster

[see web]



Foundations of Object-Oriented Languages by Kim Bruce (0 262 02523 X), MIT, 383pp @ £30-95 (1.45)
reviewed by Francis Glassborow

Do you want to understand the foundations on which object-oriented languages such C++ and Java are built? Do you want to be able to criticize design decisions that were made in developing an object-oriented language? Do you want to develop and defend proposals for change or extension to an object-oriented language? If the answer to any of these questions is 'yes' then I think you need to understand the material that is presented in this book.

This is not a book addressed directly at the practitioner though understanding the under-pinning of the languages you use may well help you to use them more effectively. For example, a clear understanding of the distinction between subclassing and subtyping together with the problems each seeks to solve will inevitably improve your design of class hierarchies.

Now one interesting feature is that the author's claimed objective is to deal with statically typed class-based object-oriented programming languages. However I feel bound to question the use of 'statically typed' in that sentence because, as I understand it, any object-oriented language must support a degree of dynamic typing to give runtime polymorphism. However I think he mainly wishes to exclude languages such as Smalltalk that are exclusively dynamically typed while including languages such as Java that allow static typing, (though not by default for user defined types). I have to admit that I struggle a little with the author's view of Java. I see its default type system as far closer to Smalltalk than to C++. It is true that Java variables are statically typed, but so are C++ pointers and references. However the underlying object (that bound to the variable in Java or reference in C++ or pointed to by a C++ pointer) has a strong dynamic element to its type. But we should note that the fact that I can enunciate points of disagreement is actually a positive aspect of the book. It makes me think and question and thereby deepen my understanding.

The book is divided into four sections. In the first section there are seven chapters that cover the fundamentals such type systems, classes, subtypes, subclasses. It concludes with a chapter describing the basics of multi-method, object-based (not exactly how I have understood the term in the past) and class based languages. Reading that made me think again about the frequent suggestion that C++ should be extended to include multi-methods. As far as this author is concerned those belong to a quite different form of object-oriented language.

The second section consists of two chapters giving an essential introduction to the lambda calculus. The advantage is that this presentation is focused on its use to model object-oriented languages. Nonetheless it is demanding and many will find the formalized expressions quite daunting.

The third section, titled 'Formal Descriptions of Object-Oriented Languages'. In fact the author develops a simple object-oriented language (SOOL) as a mechanism for communicating these ideas.

The final section consists of four chapters considering various ways of extending the language of the third section. Here we will learn about such things as bounded polymorphism, the concept of self (called MyType in this context), match-bounded polymorphism before concluding with consideration of dropping subtyping and substituting matching and hash types.

The book, overall, makes interesting reading whilst leaving me with a distinct sense that commercial interests and the requirements for backward compatibility make the application of the many lessons to real languages hard if not actually impossible. Many of us recognize that within C++ there is a simpler and yet more powerful language trying to escape and this book only serves to confirm that feeling. A good book for language lawyers and others interested in language design.

sed & awk pocket reference 2ed by Arnold Robbins (0 596 00352 8), O'Reilly, 46pp @ £8-95 (1.45)
reviewed by Francis Glassborow

[see web]

Java Server Pages Pocket Reference by Hans Bergsten (0 596 00231 9), O'Reilly, 82pp @ £8-50 (1.41)
reviewed by Francis Glassborow

[see web]

XML Processing with Perl, Python, and PHP by Martin C Brown (0 7821 4021 1), Sybex, 421pp @ £37-99 (1.32)
reviewed by Peter S Tillier

This book's cover is subtitled 'Also Covers Tcl, Rebol, Ruby and Applescript' – an awful lot for a single volume. When I received it I wondered at first if this was a good choice to ask to review, because although I have used perl and toyed with Python and Rebol, my main interest in it was to find out more about XML, its

processing and its uses. However, because I have a lot of experience using a range of scripting languages I decided that the lack of in-depth knowledge of all these languages wasn't too much of a barrier.

In fact some 250 pages are devoted to Perl, Python and PHP, with 58 pages covering the other 4 languages. The coverage of the latter four being, necessarily, short. There's an introduction to XML (56 pages) for those new to this mark-up language. I am reasonably familiar with mark-up languages so I just skimmed this introduction although I did notice a few minor errors.

I liked the way the book showed how to use SOAP (for the major languages) and XML-RPC (for all languages in the book). For some of the languages there are other XML usage examples, such as how to handle UNICODE.

The largest section of the book concentrates on python with many examples. Almost as long is the section on perl, with that on PHP occupying about half that of python. In each case there are code examples embedded in the text in a readable way. In a few examples I found the layout style slightly irritating, but this did not generally prevent me from understanding the points being made.

There are two appendices; the first discusses some UNICODE issues, while the second provides a number of references for further information.

I would recommend this book to anyone needing to use XML with a scripting language such as perl, python or PHP, but if you are using the other languages discussed you will need to do some more research using the second appendix as a starting point.

Visual Basic Design Patterns VB 6.0 & VB.NET by James Cooper (0 201 70265 7), Addison-Wesley, 484pp + CD @ £34-99 (1.29)
reviewed by Peter S Tillier

This book intends to show the reader how to implement the 23 patterns discussed in the GOF Design Patterns book using VB6 or VB.Net, which it refers to as VB7.

This book is well thought out, making use of the GOF patterns and showing how to implement examples in both versions of Visual Basic. I have used earlier versions of VB in previous jobs for my employer, so I was interested to see how they compare with the latest incarnations.

The introductory section introduces the idea of design patterns with a short history and some references and then briefly covers UML diagrams. There is not enough detail here for some, but I found the section quite useful.

The remainder of the book looks at each design pattern in turn, describes particular applications of the pattern and then provides an implementation in VB. The code examples are enhanced by well laid-out commentary, with the addition of UML or other forms of diagram where this is appropriate to clarify the points being made.

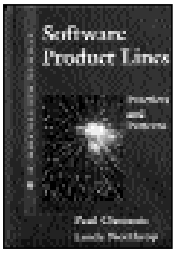
On the whole I found this book to be well written and easily readable; providing enough information to get started in using the GOF patterns. Its intended readership is clearly the VB programmer, so it may not be useful to those using other languages, hence my reservations. Even so, because VB is fairly easy to read it might be useful to more than its intended target audience.

I was not able to run the examples on the CD-ROM as I do not currently have VB6 or VB.NET installed on my machine.

Recommended with reservations.

Curl Programming Bible by Nikhil et al (0 7645 4942 1), Hungry Minds (Wiley), 755pp @ £38-99 (1.28)
reviewed by Francis Glassborow

[see web]



Software Product Lines Practices and Patterns by Paul Clements & Linda Norhrop (0 201 70332 7), Addison-Wesley, 562pp @ £37-99 (1.32)
reviewed by Huw Lloyd

The authors claim to have distilled universal and essential software product line practices that apply in every situation and to have categorised a reservoir of practical information. I believe they are correct.

The product line effort is described in three parts; software engineering, technical management and organisation management. They are further subdivided into practice areas (roles or activities); each practice is vital for the product line success. It is a simple and effective model for comprehending the whole product line effort across the business organisation. The distinction between the three categories is an eye opener in itself, one that has increasing import the further we diverge our thinking from developing a stand-alone product to a product line.

Students will benefit from the book's tractable narration and minimal technical demand. Yet the lessons and models presented are very much for the needy practitioner. The questions at every section's end are particularly good inducement for probing ones understanding.

The pattern section forms only a fraction of the book. I believe its principle benefit is for the analysis of practice area inter-relationships and comprehension and not for the claimed purpose of compiling a pattern language.

Numerous references are scattered throughout the text which are accompanied by recommendations for further reading. Notably absent was a reference for Czarnecki and Eisenecker's *Generative Programming*.

The text is well narrated, concise and full of rich content. I recommend it for anyone building multiple software products from a common source and for students of software engineering.

Recommended.

The Practical SQL Handbook 4ed by Judith Bowman et al. (0 201 70309 2), Addison-Wesley, 469pp + CD @ £34-99 (1.29)
reviewed by Joe Mc Cool

I find SQL quite easy. I think most people do. It strikes me as a simple language. After only a couple of hours even a beginner can construct some really useful queries and have data flying around all over the place!

Constructing the database is another matter. Poor design leads to poor performance, wasted space and a host of other problems. But it is not design that the current authors are concerned with, although it does get a mention. They assume, largely, that a database is already in place and their task is simply to query it.

Fair enough, the text is clear and well organised and the fact that this is fourth edition speaks volumes about the book's popularity. However there are some glaring omissions; scripting doesn't get a mention. It is as if all the code were keyed by hand at an SQL prompt. The code, it is claimed, has been tested on 'Oracle, MicroSoft, Sybase and Informix'. No mention of Unix, a single book reference to MySql! No mention of the differing environments in which SQL might run; little discussion on error reporting, security or networking; no real comparison of the SQL variants and how the called databases might differ. No mention of the possibility for programming APIs – from perl or C. Appendix E is supposed to be a full bibliography, yet Paul DuBois' classic text on MySql is nowhere to be seen. A lot of appendix space is wasted listing database data and code.

Still, for someone interested primarily in Sybase (the accompanying CD is concerned only with that) and a gentle introduction this book might be worth considering. However, I suspect it is now dated and that there are better books out there; Du Bois for one. Besides, MySql is free!

Oh and guess what, the main database example they use is one of books – titles, authors, publishers, prices – not much originality in that is there?

Non-Programming

Photoshop 6 by O'Quinn (0 7357 1130 5), New Riders, pp1070 @ £30-99 (1.29)
reviewed by Francis Glassborow

As this is not a book about programming, I am not going to take much space telling you about it. However, if you have used Photoshop you will be well aware of what a very large application it is and may well have wished for a good reference book. I think you need look no further than this book. It covers the whole of the product in 26 chapters and 13 appendices. Each Chapter deals with a single topic. For example Chapter 3 is on Vector Tools. Each chapter is split into a number of parts. For example 3.3 is one of the fifteen parts of chapter 3 and deals with the type tool. There you will find comprehensive coverage of that tool followed by issues you need to consider and then by the options available when using this tool. Every section concludes with a list of cross-references to other items in the book. That last is important because it allows you to navigate to what you really want to know even if you start a little off track.

The sad thing about this book is that while the cover has a nice tactile quality, the binding is very poor and I expect my copy to start falling apart after even moderate usage.